AD 740581

RADC-TR-72-25
Technical Report
February 1972

# A RELATIONAL MODEL OF DATA FOR THE DETERMINATION OF OPTIMUM COMPUTER STORAGE STRUCTURES

University of Michigan

UNCLASSIFIED

Security Classification

DOCUMENT CONTROL DATA - R & D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

| | |
|---|---|
| 1 ORIGINATING ACTIVITY (Corporate author)<br>Dept of Electrical Engineering<br>Systems Engineering Laboratory<br>University of Michigan, Ann Arbor, MI 48104 | 2a. REPORT SECURITY CLASSIFICATION<br>UNCLASSIFIED<br>2b GROUP |

3 REPORT TITLE

A RELATIONAL MODEL OF DATA FOR THE DETERMINATION OF OPTIMUM COMPUTER STORAGE STRUCTURES

4 DESCRIPTIVE NOTES (Type of report and inclusive dates)

Interim Report

5 AUTHOR(S) (First name, middle initial, last name)

L. Scott Randall

| | | |
|---|---|---|
| 6 REPORT DATE<br>February 1972 | 7a TOTAL NO OF PAGES<br>468 | 7b NO OF REFS<br>81 |
| 8a CONTRACT OR GRANT NO<br>F30602-69-C-0214<br>b Job Order No.<br>55810000<br>c<br>d | 9a ORIGINATOR'S REPORT NUMBER(S)<br>SEL Tech Report No. 54<br>9b OTHER REPORT NO(S) (Any other numbers that may be assigned this report)<br>RADC-TR-72-25 | |

10 DISTRIBUTION STATEMENT

Approved for public release; distribution unlimited.

| | |
|---|---|
| 11 SUPPLEMENTARY NOTES<br>RADC Project Engineer<br>Rocco F. Iuorno<br>AC 315 330-7011 | 12 SPONSORING MILITARY ACTIVITY<br>Rome Air Development Center (ISIM)<br>Griffiss Air Force Base, New York 13440 |

13 ABSTRACT

The objective of this research is the development of a rigorous quantitative method for the automatic design of optimal computer memory representations of data. A relational model of data structure is defined from which a logical ordering of data is obtained. A decision model is then developed for the specification of storage structures which can represent arbitrary data structures. Finally, a procedure is developed which determines storage structures which can represent a given data structure for which time and storage costs satisfy certain optimal conditions. A problem solution is given which demonstrates the feasibility and the effectiveness of the techniques developed.

UNCLASSIFIED
Security Classification

| 14 KEY WORDS | LINK A | | LINK B | | LINK C | |
|---|---|---|---|---|---|---|
| | ROLE | WT | ROLE | WT | ROLE | WT |
| Relational Data Structure<br>Computer Memory<br>Storage<br>Mathematical Model | | | | | | |

UNCLASSIFIED
Security Classification

# A RELATIONAL MODEL OF DATA FOR THE DETERMINATION OF OPTIMUM COMPUTER STORAGE STRUCTURES

L. Scott Randall

University of Michigan

## FOREWORD

This technical report was prepared by the Department of Electrical Engineering, Systems Engineering Laboratory, University of Michigan, Ann Arbor, MI, under contract F30602-69-C-0214, Job Order No. 55810000. Contractor's report number is SEL No. 54. The Rome Air Development Center project engineer was Rocco F. Iuorno (ISIM).

The document has been reviewed by the Office of Information (OI) and is releasable to the National Technical Information Service (NTIS).

This technical report has been reviewed and is approved.

Approved: Rocco F. Iuorno

ROCCO F. IUORNO
Info Mgt Sciences Section
Information Processing Branch

Approved: [illegible]

FRANK J. TOMAINI
Info Processing Branch
Info Sciences Div.

## ABSTRACT

In solving any problem on a digital computer one must store within the computer memory certain pieces of information, or data, upon which the program implementing the solution process makes its decisions or performs its calculations. For virtually all problems the manner in which the relationships of accessibility among the various items are portrayed by the organization of these items in the computer memory can have a marked effect upon the efficiency of the solution process. The objective of the research reported here is the development of a rigorous quantitative method for the automatic design of optimal computer memory representations for data.

To this end, we define a relational model of data structure within which we can specify the logical ordering or structure of the data involved in the solution of a given problem. We then develop a decision model for the specification of the storage structures (i.e., the computer memory representations) which can represent an arbitrary data structure.

We define two basic measures of performance - a time cost function and a storage cost function - for use in comparing the relative merits of a collection of storage structures. The time cost function reflects the number of time units required to perform certain of a set of primitive operations using a particular storage structure, and the storage cost function reflects the total number of storage units occupied by the storage structure.

Finally, we present a procedure which determines of the storage structures which can feasibly represent a given data structure the storage structure for which the time and storage costs satisfy certain optimality conditions. A storage structure is considered to be optimal if it minimizes the time cost function, subject to an upper limit on its storage cost.

To demonstrate the feasibility and the effectiveness of the techniques we develop, we apply them to a problem for which a solution program already exists and for which fairly extensive data about system performance are available. Our results demonstrate conclusively that significant improvement in program efficiency can be obtained by applying these techniques, as opposed to the historically intuitive and qualitative methods used for storage structure design.

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Continued)

## LIST OF TABLES

LIST OF FIGURES

## LIST OF SYMBOLS

This list contains in roughly chronological order the symbols which are used with some frequency in this dissertation.

| Symbol | Significance |
|---|---|
| $\Delta^o$ | The set of data items |
| $P$ | The set of relations |
| $k_r^o$ | Cardinality of $P$ |
| $\Delta$ | The set of data items used as sources |
| $k_a^o$ | Cardinality of $\Delta$ |
| $\Pi$ | The set of data items used as targets |
| $k_p^o$ | Cardinality of $\Pi$ |
| $d_i$ | An element of $\Delta$ |
| $r_j$ | An element of $P$ |
| $p_k$ | An element of $\Pi$ |
| $d_i\, r_j\, p_k$ | A relation instance |
| $\Omega$ | The set of all relation instances |
| $P_i$ | The set of relations for which $d_i \in \Delta$ is a source |
| $\Pi_\ell^2$ | A target set |
| $\Sigma_\ell$ | The set of source/relation symbol pairs for which $\Pi_\ell^2$ is the target set |
| $\Pi_m^1$ | A set in a partition of $\Pi$ |
| $\Pi_m^3$ | The set of all $\Pi_\ell^2$ of which $\Pi_m^1$ is a subset |

| Symbol | Significance |
|---|---|
| $\Gamma$ | The set of all relation symbol/target set pairs |
| $\Delta_m^2$ | A set of sources sharing common relation symbol/target set pairs |
| $\Gamma_m$ | The set of relation symbol/target set pairs shared by the elements of $\Delta_m^2$ |
| $\Delta_\ell^1$ | A set in a partition of $\Delta$ |
| $\Delta_\ell^3$ | The set of all $\Delta_m^2$ of which $\Delta_\ell^1$ is a subset |
| $\phi_i$ | A decision variable which indicates whether or not the $a_i$-ring of the SSM is an explicit ring of (forward) pointers |
| $\Delta_i$ | A decision variable which indicates whether stacking or duplication is applied to the $a_i$-ring of the SSM |
| $\phi_i'$ | A decision variable which indicates whether or not the $a_i$-ring of the SSM contains head pointers |
| $\phi_i''$ | A decision variable which indicates whether or not the $a_i$-ring of the SSM contains reverse pointers |
| $\beta_i$ | A decision variable which indicates whether or not the $b_i$-blocks of the SSM are eliminated |
| $\tau_i$ | A decision variable which indicates whether or not the $b_i$-blocks of the SSM contain type code fields |
| $\sigma_1, \sigma_2$ | Decision variables which indicate whether or not the $b_1$-blocks and the $b_{11}$-blocks of the SSM, respectively, contain description block indicators |
| $\rho_1, \rho_2, \rho_3$ | Decision variables which indicate whether or not the $b_5$-blocks, $b_6$-blocks, and $b_7$-blocks of the SSM, respectively, contain relation symbol name fields |

| Symbol | Significance |
|---|---|
| $k_i^o$ | The average cardinality of the $a_i$-rings of the SSM before transformation |
| $k_i$ | The average cardinality of the $a_i$-rings of the SSM after transformation |
| K | A matrix of products of various $k_i$ |
| $K_{ij}$ | An element of K |
| $m_a$ | The number of copies of each $b_1$-block generated by transformation of the SSM |
| $m_p$ | The number of copies of each $b_{11}$-block generated by transformation of the SSM |
| $m_{r_1}$ | The number of copies of a given $b_5$-block associated with the (copies of the) $b_1$-block representing a given source |
| $m_{r_2}$ | The number of copies of a given $b_7$-block associated with the (copies of the) $b_{11}$-block representing a given target |
| $m_i$ | The (average) total number of $b_i$-blocks in the SSM |
| $m_{r_p}$ | The expected number of times a given relation symbol is associated with a given target |
| $m_r$ | The expected number of $b_6$-blocks which represent the same relation symbol |
| $Q_i$ | A primitive operation |
| T | The time cost function |
| $a_i$ | The relative frequency of primitive operation $Q_i$ |
| $t_{ij}$ | The time cost of primitive operation $Q_i$ using method j |
| $t_i$ | The minimum of $t_{ij}$ over all j |
| $\ell_i$ | The time required to follow a forward pointer in an $a_i$-ring |

| Symbol | Significance |
|---|---|
| $s_i$ | The time required to step from one block to another in a stack for an $a_i$-ring |
| $h_i$ | The time required to follow a head pointer in an $a_i$-ring |
| $s_0$ | The time required to step from one field to another in a block |
| $e_i$ | The time cost required to move from one element block of an $a_i$-ring to another element block of that ring |
| $\delta_i$ | A binary-valued variable which indicates whether movement is toward or away from the head of an $a_i$-ring when the element blocks are stacked upon the head |
| $S_1(j, i_1)$ | A general time cost form |
| $S_2(j, i_1, i_2)$ | A general time cost form |
| $S_3(j, i_1, i_2, i_3)$ | A general time cost form |
| $S_4(j, i_1, i_2, i_3, i_4)$ | A general time cost form |
| $e_i'$ | The value of $e_i$ when $\delta_i = 0$ |
| $e_i^0$ | The time cost required to move to or from the head of an $a_i$-ring from the element block nearest the head |
| $e_i^*$ | The time cost required to move from an arbitrary element block of an $a_i$-ring to the head |
| $\hat{k}$ | The value of $(k+1)/2$ |
| $\alpha_i, \alpha_i^*$ | Sub-procedures |
| $z_i(t), z_i^*(t)$ | The time costs of $\alpha_i$ and $\alpha_i^*$, respectively |
| $f_a, f_p$ | The times required to follow a description block indicator from a $b_1$-block and from a $b_{11}$-block, respectively |

| Symbol | Significance |
|---|---|
| $F_a, F_r, F_p$ | The times required to follow a pointer in source, relation, and target rings, respectively |
| $c_d, c_r$ | The times required to compare with a given quantity a data item description and a relation symbol name, respectively |
| $v_d, v_r$ | The times required to fetch a data item description and a relation symbol name, respectively |
| $C_a, V_a$ | The times required to compare and to fetch, respectively, the data item description associated with a given $b_1$-block |
| $C_p, V_p$ | The times required to compare and to fetch, respectively, the data item description associated with a given $b_{11}$-block |
| $C_r, V_r$ | The times required to compare and to fetch, respectively, the relation symbol name associated with a given $b_6$-block |
| $C_{r_1}, V_{r_1}$ | The times required to compare and to fetch, respectively, the relation symbol name associated with a given $b_5$-block |
| $C_{r_2}, V_{r_2}$ | The times required to compare and to fetch, respectively, the relation symbol name associated with a given $b_7$-block |
| $\lambda_i$ | An inclusion variable |
| $z_i^o, z_i^{o*}$ | Expressions indicating the number of multiplies of $s_0$ required by $z_i(t)$ and $z_i^*(t)$, respectively |
| $T_a, T_r, T_p$ | The times required to find the heads of source, relation, and target rings, respectively |
| $x_i$ | A probability |
| $\delta_p$ | A binary-valued variable which indicates whether or not $m_{r_p} = 1$ |
| $m_{z_p}$ | The product $m_{r_p} m_{r_2}$ |

| Symbol | Significance |
|---|---|
| $u_i$ | The storage required for each $b_i$-block of the SSM |
| $u_d$ | The (average) storage required for a description block |
| $u_a, u_r, u_p$ | The storage required for the source, relation, and target ring heads, respectively |
| $S$ | The storage cost function |
| $s_{f_i}, s_{h_i}$ | The storage required for forward and head pointers in an $a_i$-ring, respectively |
| $s_d$ | The storage required for a description block indicator |
| $s_p$ | The storage required for source, relation, and target ring pointers |
| $s_r$ | The storage required for a relation symbol name field |
| $s_t$ | The storage required for a type code field |
| $S_o$ | The upper bound for storage available to contain the SSM |

# Chapter I

# INTRODUCTION

The major thrust of the research reported here is the development of a procedure for the automatic design of optimal schemes for the representation of data within a computer memory.

In solving any problem on a digital computer one must store in the computer memory certain pieces of information upon which the program implementing the solution process makes its decisions or performs its calculations. For most problems involving more than a few items of information the manner in which the relationships of accessibility among the various items are portrayed by the organization of these items in the computer memory can have a very marked effect upon the efficiency of the solution process. (As an extreme, imagine an hypothetical organization for which the items most frequently used are least accessible, versus an organization for which these items are most accessible.)

Clearly, we should like to organize the items of information in the computer memory in such a way as to minimize the overall cost of accessing them. On the other hand, we may also wish to minimize or to limit the amount of memory devoted to the representation of these items. Unfortunately, these two goals are often in conflict. We may be able to minimize the cost of access at the expense of the memory required to represent the items of information and vice versa, but seldom can we easily do both.

This problem is further complicated by a lack of rigorous, objective techniques for the evaluation and comparison of alternative organizational schemes, let alone any such techniques for their design.

For the most part the design of a scheme for the computer representation of data (i.e., the items of information) associated with the solution of a given problem is an art fraught with the subjectivity and the personal prejudices of the practitioner. The designer calls upon his experiences with previous systems, combines this information with a good portion of intuition and Voilà ! comes up with the perfect solution!

Indeed, this technique may produce a good solution to the problem at hand. In fact, if the designer is alert, the solution is probably better than the other alternatives which he may have considered. He still has no assurance, however, that there is no better solution to the problem. If the system in which his organizational scheme is to be imbedded will see only limited use, the designer may not care. Otherwise, he may console himself with the fact that he has done the best he can.

Perhaps we have overstated our case. The fact remains, however, that the designers of data representation systems have, at best, primitive tools with which to work.

Our objective in this research has been to develop a rigorous framework within which we can discuss the logical ordering or structure (or accessibility) of the items of information associated with a given problem, certain measures of performance with which we can compare objectively the relative merits of alternative computer representations for these items, and finally a procedure for choosing from the possible alternative representations that representation which best satisfies certain given optimality conditions.

The various sections of this chapter will be devoted to presenting some pertinent historical background, defining terms, and describing in a qualitative manner our approach to the problem.

## 1.1 A Brief History of Computer Data Representation

Due to the fact that computer memories have been organized in a linear, sequential manner since the earliest days of the stored-program computer, it is not surprising to find that all manner of data has been shaped and bent, as it were, to fit within vectors and rectangular arrays kept in consecutive memory locations.

On the other hand, neither is it surprising to find those who would rebel against the structure imposed upon their data by the inherently sequential nature of computer memories.

The first really significant break with the vector of consecutive locations was made in 1956 when Newell, Shaw, and Simon [59] introduced the concept of linked allocation and the pointer in a memory organization scheme imbedded in a list processing language (IPL) designed for heuristic problem-solving.

For their purposes Newell, Shaw, and Simon defined a list to consist of an ordered set of items of information any item of which could be another list or an element, where an element was some basic unit of information constrained to fit within a single word (i.e., location) of memory.

A list was implemented by using a set of location words, each of which was subdivided into three fields, two fields containing addresses and one field containing a type code. One address located the item corresponding to the given location word of the list and the other address located the next location word of the list. The type code in each location word simply indicated whether the item associated with that location word was a list (0) or an element (1).

Thus, if we were to define two lists A and B as

$$A = (a, B, e)$$
$$B = (b, c, d)$$

where a, b, c, d, and e are elements, then the Newell, Shaw, and Simon list structure representing these two lists would appear as in Figure 1-1, where an arrow - called a link or a pointer - originating from a given block (representing a location word) represents the address of the block, or word, to which it points.

The real innovation of this data organization lies in the fact that the addresses on the various items in a list need bear no particular relation to one another; that is, the addresses need not be consecutive. This has several implications. First, an item can be deleted from a list simply by deleting its location word. Inserting an item into a list is also facilitated. Secondly, location words on different lists (or the same one, for that matter) may contain the address of the same item of information. Finally, two or more lists may share the same region

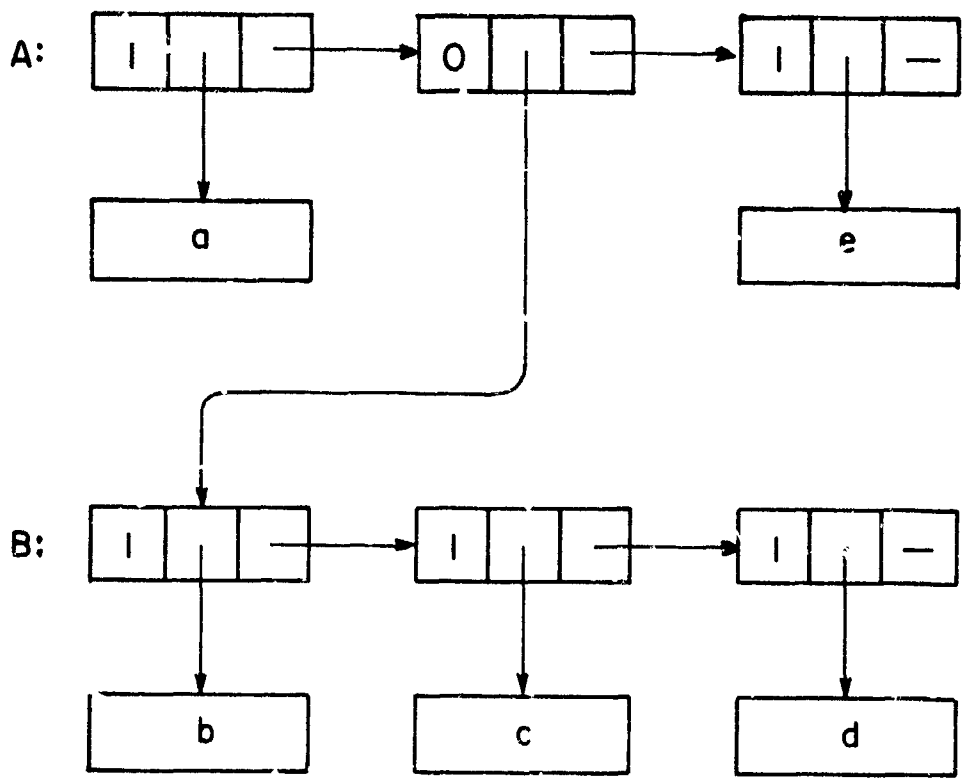


Figure 1-1. Newell, Shaw, and Simon List Structure

of memory with no list overflowing until all memory has been exhausted.

The most obvious disadvantages of the scheme are that the location words require extra memory space, with the result that approximately half of the storage used is taken up by location words, and that the ability to computer the address of the next item on a list or to determine the address of the previous item on the list is lost.

The work performed by Newell, Shaw, and Simon inspired many others to use the linked memory scheme (which at the time was often called NSS memory), and these techniques gradually evolved as basic programming tools. The first article dealing specifically with the application of linked data organizations to garden-variety problems was published by Carr [ 8 ] in 1959. In this article Carr indicated that linked lists can readily be manipulated in ordinary programming languages without the facade or restrictions of sophisticated list languages. Despite this observation, however, list processing languages continued to evolve, each with its own specific memory organization scheme.

At first one-word nodes or elements were used for the linked memory schemes, but about 1959 the usefulness of several consecutive words per node and lists structured via multiple links was being discovered. The first article dealing specifically with this idea was published by Ross [65] in 1961, and a second article dealing with multiword list items was published by Comfort [14] in 1964.

The growth in popularity of linked lists gradually produced such refinements as the circular list, or ring, in which the last item of the list contains a pointer to the first item, and the doubly linked list, in which each item of the list contains a pointer to the previous item as well as the succeeding item. The origins of these concepts cannot be attributed to specific individuals, probably because these ideas occurred naturally to many people. On the other hand, one of the main factors which lead to the widespread use of these techniques was the introduction of certain list-processing languages and systems which utilized them, in particular Weizenbaum's Symmetric List Processor (SLIP) [78,79].

Another novel concept, first introduced in 1960 by Perlis and Thornton, was that of the threaded list. In this scheme each item of a list contained three fields, one of which was a code indicating how the other two fields were to be interpreted. In particular, one of these fields contained either an item of data or a pointer to a sublist (that is, a list which functioned as an element) and the other field containe a pointer to the next item on the list or, if the item were the last on the list, a pointer to the item for which this list served as a sublist. To avoid treating as a special case the last item of any list which was not actually a sublist of any other list, a so-called head was introduced, which used the list in question as a sublist. Thus, the list

(a, (b,c), d, e)

where (b,c) acts as a sublist, would have the threaded representation shown in Figure 1-2.

Although this scheme permits rapid sequencing through levels of sublists without having to "remember" when a sublist has been encountered, it suffers the disadvantage that sublists may not be shared.

The fact is that circular lists, doubly linked lists, and threaded lists are all designed to facilitate searching or sequencing through a list and at the same time to maintain the ease of inserting and deleting items. Although additional pointers can increase somewhat the complexity of the procedures used for modification and do require additional storage above the corresponding simpler linked list, these disadvantages are often more than offset by increased ease of searching.

As we have already indicated, one of the main factors resulting

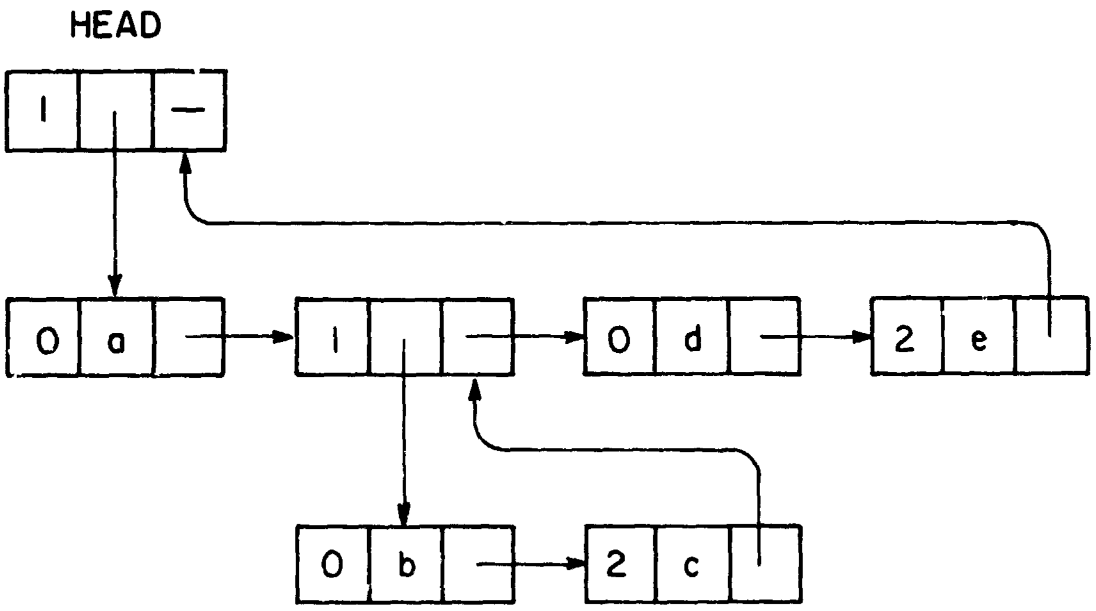


Figure 1-2. A Threaded List

in the exposure of programmers to list-processing techniques was the development and availability by the mid-1960's of a number of list processing languages and systems. The first widely used system of this sort was IPL-V [60], a relatively low-level interpretive system descended from Newell, Shaw, and Simon's IPL. Also fashioned after IPL was a system called FLPL [23], a set of FORTRAN subroutines for list manipulation developed by Gelernter. A third system called LISP [54], developed by McCarthy, involved list concepts similar to the first two but was implemented in a somewhat different manner. The most recent system of that period was Weizenbaum's SLIP [79].

As important as these systems and all others of their type were (and still are!), they all suffer from one major flaw, the same flaw which characterized the vector of consecutive locations a which in fact lead to their development: inflexibility. Each of these systems provides a fixed type of structure to which the data to be represented must be tailored. Ideally the memory representation should conform to the data and the application at hand.

In 1966, however, Knowlton introduced a general, relatively low-level, list processing system called $L^6$ [41,42], the features of which were quickly incorporated into a number of other systems. $L^6$ allows the programmer to choose blocks of sizes to his liking, to define fields within these blocks as he sees fit, and in general to manipulate his structures as he desires. While not a panacea, $L^6$ is certainly a step in the right direction.

Dissatifaction with certain aspects of linked memory schemes (in addition to dissatisfaction with the sequential schemes) when applied to certain types of problems caused some individuals to seek still further alternatives. In particular, Feldman [20] and Rovner [67] were concerned with problems in the area of artificial intelligence for which an associative memory, in which reference to information stored in the memory is made by specifying the contents of a part of a cell (word) instead of an address, would be decidedly advantageous. Unfortunately, the cost of a hardware memory of this type and of the size required for such problems was prohibitive. Therefore, in 1965 Feldman proposed his version of a software-simulated associative memory for a computer with conventional memory and introduced the concept of hash-coding as a form of address determination.

The basic idea behind hash-code addressing is that given the contents of some part of a memory cell (where a cell may consist of more than one word), the address of that cell can be determined by applying some transformation to the contents given.

In Feldman's scheme each cell contains three principal fields which we will designate $F_1$, $F_2$, and $F_3$, the contents of which are presumably related to one another in some fashion. A triple consisting of a value for each of the fields $F_1$, $F_2$, and $F_3$ is assigned an address in the memory by "hash-coding" the values for fields $F_1$ and $F_2$. In Feldman's case the hash-coding is implemented by shifting the value for field $F_1$ left a number of places, performing a partial add between this result and the value for field $F_2$, and insuring that the result is even.

Retrieving the value of field $F_3$ given the values of fields $F_1$ and $F_2$ then simply involves hashing the values of $F_1$ and $F_2$ to determine the address of the cell containing the value sought.

Obviously, there are a number of problems which can arise in using such a scheme. In the first place there may be more than one value for $F_3$ associated with a given pair of values for $F_1$ and $F_2$. This situation is called <u>multiplicity</u>. Secondly, it is possible that two distinct pairs of values for $F_1$ and $F_2$ may hash to the same address. This situation is called <u>overlap.</u> Finally, the cell accessed by hashing a particular pair of values for $F_1$ and $F_2$ may have been used as a free-storage cell to handle the overlap or multiplicity of another cell. This situation is called <u>conflict</u> and requires moving the present contents of that cell, a very costly operation.

Feldman handles the overlap and multiplicity problems - and provides for such operations as determining the values of $F_1$ and $F_2$ given the value of $F_3$ - by including certain link fields in each cell.

About a year after Feldman introduced his version of an associative processor, Rovner proposed certain extensions to the scheme to make it feasible for such a system to be used in the paged environment of a virtual memory machine. The basic concepts, however, remained unchanged.

From this point on in time the number of memory representations has grown very rapidly. The principles involved in each of these "new" representations are, however, basically no different from those we have presented so far.

In a recently published text, Knuth [43] presents the basic principles of memory representations in a very readable form. In this reference the various fundamental memory representations are divorced from all languages and systems in which they have been previously imbedded and are considered on their own merits. For instance, rather than discussing how a given situation might be handled using SLIP, Knuth considers how doubly linked lists (basically the structures generated by SLIP) might be utilized.

In this section we have attempted to present some of the highlights in the evolution of computer memory data representations. The reader should not expect this to be an exhaustive consideration of all existing organizational schemes, for it surely is not. Neither should the reader expect this to be a tutorial on the basic principles of computer memory representations. For such a discussion the reader is referred to the text by Knuth. This section is intended primarily to give the reader a flavor for the problems and considerations involved in the design of data representations for the computer memory.

## 1.2 Definitions and Philosophy

Up to this point our discussions, out of necessity, have been of a rather intuitive and subjective nature. We have used such imprecise terms as "computer memory representation" and "data organization scheme" in an attempt to describe the problem with which we are concerned. In this section we intend to define a number of terms so that our future discussions may assume a less ambiguous posture.

We should note at the outset that the individuals (and the corresponding literature) concerned with computer memory data organizations may generally be grouped into two rather ill-defined and overlapping classes. On the one hand are those individuals who are primarily concerned with the representation of data within the main store (e.g., core memory) of a computer, and on the other hand are those who are primarily concerned with the organization of data within the secondary store (e.g., drum and disc) of a computer or within a hierarchy of memory devices. The former might be called "data structure* enthusiasts" and the latter, "file management enthusiasts".

* The term "data structure" is used here in a very broad, generic sense which differs from our later definition of the term.

Fundamentally, the problems which face these two groups are identical. There are differences, however, in the respective environments in which their solutions to these problems must operate. The inherently sequential nature of accessing information on a disc may dictate a different organizational scheme than the random access nature of core memory.

We do not wish to dwell upon these differences (for the similarities may more than outweigh them) but merely wish to indicate that there are frequently (but not always) differences in the terminology used by the two groups. For example, whereas the data structure people are generally concerned with the interrelationships among <u>data items</u> or <u>data elements</u>, the file management people are generally concerned with the interrelationships among <u>files</u> and <u>records</u>.

For the purposes of this exposition we will reside in the camp of the data structure people.

Let us now proceed with the definition of some terms. In solving a particular problem on a digital computer, the program which implements the solution process operates upon some collection of objects which are used as the basis for decision or calculation. Each of these objects represents an occurrence, or an instance, of some physical or conceptual quantity and is characterized by an ordered pair consisting of a <u>data name</u> and a <u>data value</u>. The data name indicates the quantity itself, such as planet, river, city, etc., and the data value indicates the particular instance of this quantity.

The data name/data value ordered pair (or, equivalently, the object characterized by the ordered pair) will be called a data item. A data item is denoted by its corresponding data name, and an instance of a data item is a specific data name/ data value pair. For example, the ordered pair (CITY, ANN ARBOR) is an instance of the data item CITY.

The definitions of data name, data value, and data item which we have just presented are essentially the same as the definitions used by McCuskey[55], who points out that in common high-level programming language usage the data value corresponds to the "data" which is stored in the computer memory while the data name corresponds to "data about data" which appears in the source program and enters a symbol table during compilation.

The data name and the data value are represented in the computer memory by sequences of elementary objects called symbols which are members of some alphabet, or set of all symbols, and which are known to the program. One possible alphabet is the EBCDIC character set.

The ordered pair of symbol sequences which represent a data name/ data value pair is called a data item description, or simply a description.

It should be clear that a given data item may have any of a number of descriptions depending upon the alphabet chosen.

Let us now examine the actual steps involved in solving a problem on a digital computer.

First, of course, we must define the problem to be solved. This involves specifying such things as the information (i.e., the data items) which the solution process is to be given initially and upon which it is to operate, the algorithms (perhaps in the form of flow charts) which are pertinent to the solution process, and finally the results which the solution process is expected to determine. These specifications must be complete and concise and must reflect the requirements of any solution of the problem, manual or automatic.

Second, we must specify the logical ordering or structure of the various data items - that is, the logical relationships of accessibility among the various data items - and the logical access processes which may be used to find any data item in the structure. The result of this step is a specification of the data structure of the data items involved in the solution of the problem.

Third, given the specification of the data structure for the problem, we must determine a suitable physical organization for the data items within the computer memory. The resulting representation is called the storage structure of the data.

Finally, we must actually generate the code for the program required to implement the solution process and the specified storage structure.

For the purpose of this exposition we will concern ourselves primarily with the second and third of these steps. We assume that we are given the problem definition as an initial starting point and that we produce the specifications used by the system implementor.

The reader should note the distinction we have made between the terms "data structure" and "storage structure". This distinction was first made by D'Imperio [17] in 1964 and later by Mealy [57] in 1967. The reader is cautioned, however, that not all authors use these terms in the same way. In fact, it is frequently the case that the term data structure is used to encompass the spectrum of both data structure and storage structure as we have defined them.

In the context of our definitions, data structures arise from the interpretations we give to certain aspects of a problem and its solution, and they are not necessarily invariant, inherent, or necessary characteristics of the data items themselves.

To contrast the concepts of data structure and storage structure once again we might say that data structures are simply theories of the structure of the real world, and storage structures are computer representations of these theories.

It is patently clear that even for the simplest of data structures there are a multitude of storage structures which may be used to represent them. Consider for instance a collection of n data items $d_i$, where $i \in \{1, 2, \cdots, n\}$, for which the corresponding data structure is linear and sequential in nature, such that data item $d_1$ is the first in the sequence, data item $d_n$ is the last in the sequence, and data item $d_i$ is preceeded by data item $d_{i-1}$ and followed by data item $d_{i+1}$. Some of the more obvious choices of storage structures which may be used to represent this data structure include 1) a vector of consecutive memory locations, 2) a linked linear list, and 3) a doubly linked list.

Herein, of course, lies our problem: determine the storage structure which best represents a given data structure. Before considering our approach to the problem, however, let us make some general observations concerning storage structures.

All storage structures may be classified according to one or more of three basic types of organization: <u>sequential</u>, <u>list</u>, and <u>random</u>. These types, to which we alluded in the previous section, are described by Dodd* [19], so our consideration of them will be relatively brief.

In sequential organization data items are stored in the computer memory in locations relative to other data items according to a

---

* Note that Dodd's treatment of the subject is from the viewpoint of file management and his definition of data structure corresponds to our definition of storage structure.

specified sequence. Most commonly, the data items are stored in consecutive locations of the memory. We could, however, use any other well defined sequence. For instance, the data items could be stored in locations whose addresses are given by increasing relatively prime numbers (although the utility of such a scheme might be questionable).

The basic concept of list organization is that pointers are used to separate the logical ordering of the memory locations containing data items from the physical ordering of these locations. In general, a pointer may be anything which allows the accessing mechanism to locate the memory cell containing a given data item, but almost without exception a pointer is considered to be a value representing the address of the cell containing the data item.

Finally, in random organization data items are stored and retrieved on the basis of some predictable relationship between the data item and the address of its assigned memory location. We may distinguish three basic types within random organization: direct address, dictionary look-up, and calculation.

For the direct address method of random organization an arbitrary absolute address is assigned to a data item by the programmer and this address is used every time the data item is to be accessed.

The dictionary look-up method involves maintaining a dictionary or symbol table containing pairs of keys and addresses for the data items of interest. To determine the location of a given data item, the dictionary is searched for the key (perhaps the data name) associated with the data item; the address associated with this key indicates the location of the desired data item.

Lastly, the calculation method converts the key of a given data item into an address (not necessarily unique) by performing a standard calculation or transformation upon it. This method includes the hash-coding scheme we discussed in the previous section.

We wish to reiterate the fact that most storage structure schemes are not purely sequential, list, or random in nature, but utilize the properties of all of these schemes to lesser or greater extents.

In later discourse we will desire to make frequent reference to a particular class of sequentially organized storage structures - namely, those storage structures for which the data items are stored in consecutive cells of memory. To facilitate our discussions, let us define such storage structures to be stacks*. To be more specific, let a stack be defined to consist of a vector, or block, of consecutive memory locations which contains a number of data items ordered in the same manner as the memory locations which contain them.

Let us now conclude our discussion of the philosophy of data representation by summarizing its two most important points. First, the concepts of data structure and storage structure are distinct.

* Note that this definition of the term stack is not necessarily a standard definition and differs, in particular, from the definition used by Knuth [ 43 ].

Data structure refers to the logical ordering or structure of data as we interpret it for the solution of a given problem. Storage structure refers to the physical representation of data structure within a computer memory. Second, a given data structure can invariably be represented by a number of distinct storage structures

## 1.3 Research Objectives

It is evident that determination of a storage structure to represent the data structure associated with the solution of a given problem is a process involving a large number of tradeoffs. By now the qualitative implications of these tradeoffs are relatively well understood. What is needed, however, are rigorous, objective techniques for evaluation of the quantitative aspects of the tradeoffs. The result of developing such techniques should be more intelligent systems design, leading in turn to programs operating with higher productivity at a lower cost.

Our approach to development of these techniques is threefold. First, we will develop a rigorous framework in the form of a relational and set-theoretic model for the description of data structure. The relational view of data, which has been advocated by others (in particular, Childs [11] , Codd [12] , McCuskey[55] , and Mealy [57] ) appears to be superior in several respects to a graph or network model since it does not superimpose any structure upon the data which might fall within the realm of storage structure.

Second, we will develop a decision model for specifying the storage structures capable of representing any data structure as given by our data structure model.

Finally, we will develop certain measures of performance which enable us to compare the time and storage characteristics of the storage structures described by our storage structure model. We will also present a procedure for examining the measures of performance for each of the set of storage structures which can feasibly represent a given data structure and for determining that storage structure for which these measures of performance best satisfy certain optimality conditions.

# Chapter II

# A MATHEMATICAL MODEL OF DATA STRUCTURE

In order to address ourselves to the problem of choosing an optimal storage structure for a particular collection of data, we need a rigorous framework within which we can discuss the structure of data.

The subject of this chapter then is the specification of a mathematical model, or abstraction, of data structure.

## 2.1 Relations

Since, as we have indicated, our model of data structure will be a relational model, we devote this section to defining the concept of a relation.

A propositional function defined on the Cartesian product $A \times B$ of two sets A and B is an expression denoted by $P(x,y)$ which has the property that $P(a,b)$, where a and b are sustituted for the variables x and y in $P(x,y)$, is true or false for any ordered pair $(a,b) \in A \times B$.

For example, if A is the set of all composers and B is the set of all musical compositions, then

P(x,y) = "x composed by y"

is a propositional function on $A \times B$. In particular,

P(Berlioz, Symphonie Fantastique)
= "Berlioz composed Symphonie Fantastique"

and

P(Bach, 1812 Overture)
= "Bach composed 1812 Overture"

are true and false, respectively.

The expression $P(x,y)$ by itself is called an open sentence in two variables or, simply, an open sentence.

A <u>relation</u> r may be defined to consist of the following:

(1) a set A

(2) a set B

(3) an open sentence P(x,y) in which P(a,b) is either true or false for any ordered pair $(a,b) \in A \times B$.

Thus, r is called a <u>relation from A to B</u> which we will denote by

$$r = (A, B, P(x,y))$$

Furthermore, if P(a,b) is true, we will denote this fact by

$$a\ r\ b$$

and if P(a,b) is not true, we will denote this fact by

$$a \not r\ b$$

Let r = (A,B,P(x,y)) be a relation. We define the <u>solution set</u> R of the relation r to consist of the elements (a,b) in $A \times B$ for which P(a,b) is true. That is,

$$R = \{(a,b) \mid a \in A,\ b \in B,\ P(a,b) \text{ is true}\}$$

Notice that R, the solution set of relation r from A to B, is a subset of $A \times B$.

Let R be any subset of $A \times B$. Then we can define a relation r = (A, B, P(x,y)) where P(x,y) = "The ordered pair (x,y) belongs to R". The solution set of this relation r is the original set R.

Thus, to every relation $r = (A, B, P(x,y))$ there corresponds a unique solution set $R$ which is a subset of $A \times B$, and to every subset $R$ of $A \times B$ there corresponds a relation $r = (A, B, P(x,y))$ for which $R$ is the solution set. Since this one-to-one correspondence exists between relations $r = (A, B, P(x,y))$ and subsets $R$ of $A \times B$, we can redefine a relation as follows:

> A relation $r$ from $A$ to $B$ is a subset of $A \times B$.

Although this definition may appear somewhat artificial, it has the advantage that the undefined concepts of "open-sentence" and "variable" are not used.

## 2.2 A Relational Model of Data Structure

In this section we will present a rigorous, mathematical model for data structure. In the course of doing so we will define a rather large number of sets. Since our motive for defining a particular set may not present itself until other sets have been defined, we caution the reader of this possibility ahead of time.

We raise one further word of caution: Several distinct sets may be designated by the same symbol (a capital Greek letter) with a superscript used to differentiate among them. In general, sets designated by a common symbol and distinguished in this manner will share some common (subjective) properties. Thus, no special significance other than its role as a device for differentiation should be attached to the value of a superscript.

Let us now proceed with the development of our model for data structure.

The intrinsic structure, or the data structure, of any collection of n data items may be described in the following manner.

Let the set $\Delta^0$ consist of the n data items in question:

$$\Delta^0 = \{d_i \mid i = 1, 2, \cdots, n\}$$

where $d_i$ is the i-th data item in the collection.

Let the set $P$ (capital rho) consist of all relations of interest in $\Delta^o$ (i.e., from $\Delta^o$ to $\Delta^o$):

$$P = \{r_j \mid j = 1, 2, \cdots, k_r^o\}$$

where $r_j$ is the j-th relation and $k_r^o$ is the number of different relations of interest.

When we refer to the "relations of interest", we mean that for the purpose of solving a particular problem involving the data items of $\Delta^o$ we may assume that the elements ot $\Delta^o$ are related to one another only via the relations in $P$ (whereas these data items may actually be related to one another via other relations as well).

The set $\Delta^o$ and the relations o. $P$ then define the intrinsic structure of the given collection of n data items.

This information in itself is of little more than academic interest, but we may expand upon it somewhat in order to gain the insight required of our model.

Let $R_j$ be the solution set of relation $r_j \in P$. $R_j$ then consists of a set of ordered pairs $(d_i, d_k)$ in $\Delta^o \times \Delta^o$.

Let us define $\Delta_j$ to be a subset of $\Delta^o$ such that every element of $\Delta_j$ is the first element of at least one ordered pair $(d_i, d_k)$ in $R_j$.

$$\Delta_j = \{d_i \mid d_i \in \Delta^o, (d_i, d_k) \in R_j \text{ for some } d_k \in \Delta^o\}$$

Further, let us define $\Pi_j$ to be a subset of $\Delta^o$ such that every element of $\Pi_j$ is the second element of at least one ordered pair $(d_i, d_k)$ in $R_j$.

$$\Pi_j = \{d_k \mid d_k \epsilon \Delta^o, (d_i, d_k) \epsilon R_j \text{ for some } d_i \epsilon \Delta^o\}$$

Let

$$\Delta = \bigcup_{j=1}^{k_r^o} \Delta_j$$

and let

$$\Pi = \bigcup_{j=1}^{k_r^o} \Pi_j$$

Clearly $\Delta$ and $\Pi$ are subsets of $\Delta^o$. In fact either $\Delta$ or $\Pi$ or both may be identically equal to $\Delta^o$, but this is not necessarily the case (which is, of course, our reason for defining $\Delta$ and $\Pi$).

The set P may now be defined to consist of relations from $\Delta$ to $\Pi$. The elements of P are exactly the same as before.

As a practical matter we will require that $\Delta^o = \Delta \cup \Pi$. That is, we will require every element of $\Delta^o$ to be related by at least one relation in P to at least one other element in $\Delta^o$ (not excluding the possibility of an element being related to itself). Since we are concerned with the interrelationships among various data items in our consideration of data structure, we find a data item which is related to no other to be singularly uninteresting. (There are, of course, some interesting problems associated with the representation of elementary items of data within a computer - such as, the representation of rational numbers - but these problems do not concern us here.)

Let the cardinality of set $\Delta$ (which we denote by $|\Delta|$) be $k_a^o$ and let the cardinality of set $\Pi$ be $k_p^o$.

$$|\Delta| = k_a^o$$

$$|\Pi| = k_p^o$$

As a matter of notational convenience, let the k-th element of $\Pi$ be denoted by $p_k$ , where $k \epsilon \{1,2,\cdots,k_p^o\}$ , to distinguish it from the k-th element of $\Delta$ which we will continue to denote by $d_k$, where $k \epsilon \{1,2,\cdots,k_a^o\}$.

We then indicate the fact that some data item $d_i$ in $\Delta$ is related to some data item $p_k$ in $\Pi$ via some relation $r_j$ in $P$ by the notation $d_i \; r_j \; p_l$ . For specific values of i, j, and k we call $d_i \; r_j \; p_k$ a <u>relation instance</u>. In particular, $d_i \; r_j \; p_k$ is a relation instance if $(d_i, p_k) \epsilon R_j$. $d_i$ is called the <u>source</u> of the relation instance, $p_k$ is called the <u>target</u> of the relation instance, and $r_j$ is called the <u>relation symbol</u> of the relation instance. Note that we may use $r_j$ to denote both a relation and a relation symbol. Context should make clear, however, the sense in which $r_j$ is being used.

We now define a set $\Omega$ to contain all the relation instances implied by the relations in P.

$$\Omega = \{d_i \, r_j \, p_k \mid (d_i, p_k) \in R_j, \; j = 1, 2, \cdots, k_r^o\}$$

Corresponding to each data item $d_i \in \Delta$, where $i \in \{1, 2, \cdots, k_a^o\}$, there exists some (nonempty) set $P_i$ of relations such that

(1) $P_i \subseteq P$, and

(2) for each relation $r_j \in P_i$ there exists at least one data item $p_k \in \Pi$, where $k \in \{1, 2, \cdots, k_p^o\}$, for which $d_i \, r_j \, p_k$.

$$P_i = \{r_j \mid r_j \in P, \; d_i \in \Delta, \; (d_i, p_k) \in R_j \text{ for some } p_k \in \Pi\}$$

In other words, $P_i$ consists of all relations in P whose solution sets contain at least one ordered pair the first element of which is $d_i$.

Corresponding to every source/relation symbol pair $(d_i \, r_j)$, where $d_i \in \Delta$ and $r_j \in P_i \subseteq P$, there exists some set $\Pi_\ell^2 \subseteq \Pi$ of targets, where $\ell \in \{1, 2, \cdots, n_2\}$, such that for every target $p_k \in \Pi_\ell^2$, $(d_i, p_k) \in R_j$. That is, $\Pi_\ell^2$, which is called the $\ell$-th target set, consists of all data items $p_k \in \Pi$ which satisfy $d_i \, r_j \, p_k$.

To present a more formal definition, we define the target set $\Pi_\ell^2$ for $\ell \in \{1, 2, \cdots, n_2\}$ such that

(1) $\Pi^2_\ell \subset \Pi$

(2) $\Pi^2_\ell \neq \emptyset$

(3) $\Pi^2_\ell = \{p_k \mid d_i\, r_j\, p_k \in \Omega$ for fixed $d_i$ and $r_j\}$

(4) $\Pi^2_{\ell_1} \neq \Pi^2_{\ell_2}$ if $\ell_1 \neq \ell_2$

Note that this definition does not preclude the possibility of two or more distinct source/relation symbol pairs having the same target set.

Let the set $\Sigma_\ell$, where $\ell \in \{1, 2, \cdots, n_2\}$, consist of all source/relation symbol pairs $(d_i\, r_j)$, where $d_i \in \Delta$ and $r_j \in P_i$, such that $(d_i, p_k) \in R_j$ for every $p_k \in \Pi^2_\ell$.

$$\Sigma_\ell = \{(d_i\, r_j) \mid d_i\, r_j\, p_k \in \Omega \text{ for all } p_k \in \Pi^2_\ell\}$$

That is, $\Sigma_\ell$ consists of all source/relation symbol pairs which have $\Pi^2_\ell$ as target set. We note that there is a one-to-one correspondence between the elements of the set $\{\Sigma_\ell \mid \ell = 1, 2, \cdots n_2\}$ and the elements of the set $\{\Pi^2_\ell \mid \ell = 1, 2, \cdots n_2\}$. For convenience we have assumed that $\Sigma_\ell$ corresponds to $\Pi^2_\ell$ for all $\ell \in \{1, 2, \cdots, n_2\}$.

Let the set $\Pi$ be partitioned into a number $n_1 \leq k^o_p$ of disjoint subsets $\Pi^1_m$, where $m = 1, 2, \cdots n_1$, such that

(1) $\Pi^1_{m_1} \cap \Pi^1_{m_2} = \emptyset$ for $m_1 \neq m_2$

(2) $\Pi = \bigcup_{m=1}^{n_1} \Pi^1_m$

$$(3)\quad p_{k_1}, p_{k_2} \in \Pi^1_m \Longleftrightarrow$$

$$(p_{k_1} \in \Pi^2_\ell \Longleftrightarrow p_{k_2} \in \Pi^2_\ell)$$

$$\text{for all } \ell \in \{1, 2, \cdots n_2\}$$

The main thrust of this definition, as contained in condition 3, is the following. If targets $p_{k_1}$ and $p_{k_2}$ are both elements of the set $\Pi^1_m$, then for every set $\Pi^2_{\ell_1}$ of which $p_{k_1}$ is an element, $p_{k_2}$ is also an element, and for every set $\Pi^2_{\ell_2}$ of which $p_{k_2}$ is an element, $p_{k_1}$ is also an element. Conversely, if for every set $\Pi^2_{\ell_1}$ of which $p_{k_1}$ is an element $p_{k_2}$ is also an element and if for every set $\Pi^2_{\ell_2}$ of which $p_{k_2}$ is an element $p_{k_1}$ is also an element, then $p_{k_1}$ and $p_{k_2}$ are both elements of the same set $\Pi^1_m$.

We note that for each set $\Pi^2_\ell$, where $\ell \in \{1, 2, \cdots, n_2\}$, there exists a set $M_\ell$ of indices such that

$$M_\ell \subset \{1, 2, \cdots, n_1\}$$

$$\Pi^2_\ell = \bigcup_{m \in M_\ell} \Pi^1_m$$

On an intuitive basis we can view the sets $\Pi^1_m$ as the largest subsets into which $\Pi$ can be partitioned while still allowing the construction of each $\Pi^2_\ell$ as a union of some of them. Obviously, we can partition $\Pi$ into $k^o_p$ subsets, each of which contains exactly one data item.

Suppose, however, that for every target set $\Pi^2_\ell$ in which data item $p_{k_1} \epsilon \Pi$ appears, data item $p_{k_2} \epsilon \Pi$ also appears. Then $p_{k_1}$ and $p_{k_2}$ can be combined into a single set, thereby reducing the number of subsets $\Pi^1_m$ to $k^o_p$-1. Continuing in this manner we eventually reach the point at which no two targets in $\Pi$ appear together in any target set $\Pi^2_\ell$ in which one or the other appears but are not elements of the same subset of the current partition. At this point we cannot further reduce the number of subsets in the partition and these subsets define the various $\Pi^1_m$.

Let the set $\Pi^3_m$, where $m \epsilon \{1, 2, \cdots n_1\}$, consist of all those target sets $\Pi^2_\ell$ of which $\Pi^1_m$ is a subset.

$$\Pi^3_m = \{\Pi^2_\ell \mid \Pi^1_m \subset \Pi^2_\ell\}$$

$\Pi^3_m$ simply indicates which of the target sets $\Pi^2_\ell$ "use" the subset $\Pi^1_m$.

Given some data item $d_i \epsilon \Delta$, for every $r_j \epsilon P_i$ there exists some corresponding target set $\Pi^2_\ell$ (such that $(d_i, p_k) \epsilon R_j$ for each $p_k \epsilon \Pi^2_\ell$). Let the set $\Gamma$ consist of all distinct relation symbol/target set pairs $(r_j \Pi^2_\ell)$ which are associated with all sources in $\Delta$ such that

(1) $\Gamma = \{(r_j \Pi^2_\ell) \mid d_i r_j \Pi^2_\ell \epsilon \Omega \text{ for some } d_i \epsilon \Delta\}$

(2) $(r_j \Pi^2_{\ell_1}), (r_j \Pi^2_{\ell_2}) \epsilon \Gamma, \Pi^2_{\ell_1} \subset \Pi^2_{\ell_2}, \ell_1 \neq \ell_2$

$\exists\ d_{i_1}, d_{i_2} \epsilon \Delta, i_1 \neq i_2$ such that $d_{i_1} r_j \Pi^2_{\ell_1}, d_{i_2} r_j \Pi^2_{\ell_2} \epsilon \Omega$

where $d_i\ r_j\ \Pi^2_\ell$ represents all relation instances $d_i\ r_j\ p_k$ such that $p_k \epsilon\ \Pi^2_\ell$. Clearly, $|\Gamma| \leq k^o_a k^o_r$.

It may be the case, of course, that the same relation symbol/target set pair is associated with more than one source. Let $\Delta^2_q$ be the set of all sources in $\Delta$ with which the q-th element $(r_j\ \Pi^2_\ell)$ of $\Gamma$ is associated, where $q \epsilon \{1, 2, \cdots, |\Gamma|\}$.

Let us define a one-to-one function $\sigma$ which assigns to each element of $\Gamma$ a unique element of the set $\{1, 2, \cdots, |\Gamma|\}$.

$$\sigma : \Gamma \rightarrow \{1, 2, \cdots, |\Gamma|\}$$

(Clearly, $\sigma$ maps $\Gamma$ <u>onto</u> the set $\{1, 2, \cdots |\Gamma|\}$.) We may use the function $\sigma$ to assign an index value q to each element $(r_j \Pi^2_\ell)$ of $\Gamma$:

$$q = \sigma(r_j \Pi^2_\ell)$$

Similarly, we may use the inverse $\sigma^{-1}$ of the function $\sigma$ to determine the element $(r_j \Pi^2_\ell)$ of $\Gamma$ corresponding to an index value q:

$$(r_j \Pi^2_\ell) = \sigma^{-1}(q)$$

Then the set $\Delta^2_q$ may be defined more rigorously as follows:

$$\Delta^2_q = \{d_i \mid d_i \epsilon\ \Delta,\ d_i r_j\ \Pi^2_\ell \epsilon\ \Omega,\ (r_j \Pi^2_\ell) = \sigma^{-1}(q)\}$$

where $q \epsilon \{1, 2, \cdots, |\Gamma|\}$.

Let

$$\Lambda = \{\Delta_q^2 \mid q = 1, 2, \cdots, |\Gamma|\}$$

There may exist sets $\Delta_{q_1}^2$ and $\Delta_{q_2}^2 \epsilon \Lambda$ such that $\Delta_{q_1}^2 = \Delta_{q_2}^2$ for $q_1 \neq q_2$.

Therefore, let us partition the set $\Lambda$ into a number $n_4 \leq |\Gamma|$ of subsets, each of which consists of a collection of identically equal elements. Let us then choose one element from each of the subsets to form a new set $\Lambda^*$ of distinct elements. In particular, let us define the set $\Lambda^*$ as follows:

(1) $\Lambda^* = \{\Delta_m^2 \mid \Delta_m^2 \epsilon \Lambda\}$

(2) $\Delta_{m_1}^2 \neq \Delta_{m_2}^2$ for $\Delta_{m_1}^2, \Delta_{m_2}^2 \epsilon \Lambda^*$ and $m_1 \neq m_2$

We note that $|\Lambda^*| = n_4$.

For each $\Delta_m^2 \epsilon \Lambda^*$ we may then define a set $\Gamma_m \subset \Gamma$ which consists of all relation symbol/target set pairs associated with all $\Delta_q^2 \epsilon \Lambda$ for which $\Delta_q^2 = \Delta_m^2$. Specifically, we define the set $\Gamma_m$ for $m \epsilon \{1, 2, \cdots, n_4\}$ su[illegible] that

(1) $\Gamma_m \neq \emptyset$

(2) $\Gamma_m = \{(r_j \Pi_\ell^2) \mid (r_j \Pi_\ell^2) \epsilon \Gamma, d_i r_j \Pi_\ell^2 \epsilon \Omega$ for all $d_i \epsilon \Delta_m^2 \epsilon \Lambda^*\}$

(3) $\Gamma_{m_1} \neq \Gamma_{m_2}$ if $m_1 \neq m_2$

Now in the same manner as we did for $\Pi$, let us partition the set $\Delta$ into a number $n_3 \leq k_a^o$ of disjoint subsets $\Delta_\ell^1$, where $\ell = 1, 2, \cdots n_3$, such that

(1) $\Delta_{\ell_1}^1 \cap \Delta_{\ell_2}^1 = \emptyset$ for $\ell_1 \neq \ell_2$

(2) $\Delta = \bigcup_{\ell=1}^{n_3} \Delta_\ell^1$

(3) $d_{i_1}, d_{i_2} \in \Delta_\ell^1 \Rightarrow$

$$(d_{i_1} \in \Delta_m^2 \Leftrightarrow d_{i_2} \in \Delta_m^2)$$

for all $m \in \{1, 2, \cdots, n_4\}$

(i.e., for all $\Delta_m^2 \in \Lambda^*$)

It follows that for each set $\Delta_m^2 \in \Lambda^*$, where $m \in \{1, 2, \cdots, n_4\}$, there exists a set $L_m$ of indices such that

$$L_m \subset \{1, 2, \cdots, n_3\}$$

$$\Delta_m^2 = \bigcup_{\ell \in L_m} \Delta_\ell^1$$

Finally, let the set $\Delta_\ell^3$, where $\ell \in \{1, 2, \cdots, n_3\}$, consist of all those sets $\Delta_m^2 \in \Lambda^*$ of which $\Delta_\ell^1$ is a subset.

$$\Delta_\ell^3 = \{\Delta_m^2 \mid \Delta_m^2 \in \Lambda^*, \Delta_\ell^1 \subset \Delta_m^2\}$$

$\Delta_\ell^3$ simply indicates which of the sets $\Delta_m^2$ "use" the subset $\Delta_\ell^1$.

### 2.2.1 Schematic Representation

Our model for the data structure of an arbitrary collection of elementary items of data now consists of the sets $\Delta$, $\Pi$, $P_i$, $\Delta_\ell^1$, $\Delta_\ell^3$, $\Delta_m^2$, $\Gamma_m$, $\Sigma_\ell$, $\Pi_\ell^2$, $\Pi_m^3$, and $\Pi_m^1$ as well as our original sets $\Delta^0$ and $P$.

In order to clarify the relationships among these sets, we introduce the schematic representation of Figure 2-1. We may view this structure as an undirected graph consisting of certain nodes, or vertices, connected by (undirected) branches, or edges.

Each ring of the structure of Figure 2-1 contains some number $k^0 + 1$ of nodes (where the value of $k^0$ varies from ring to ring). Of these $k^0 + 1$ nodes $k^0$ act in a "connective" capacity and the remaining node represents some set. In particular, for any ring in the structure those nodes of the ring which lie on a level designated by an odd number are the connective nodes and that node which lies on a level designated by an even number is the set node. In a general sense, we may interpret the connective nodes of a ring as indicating the elements of the set represented by the set node of the ring.

Before proceeding with the interpretation of the structure of

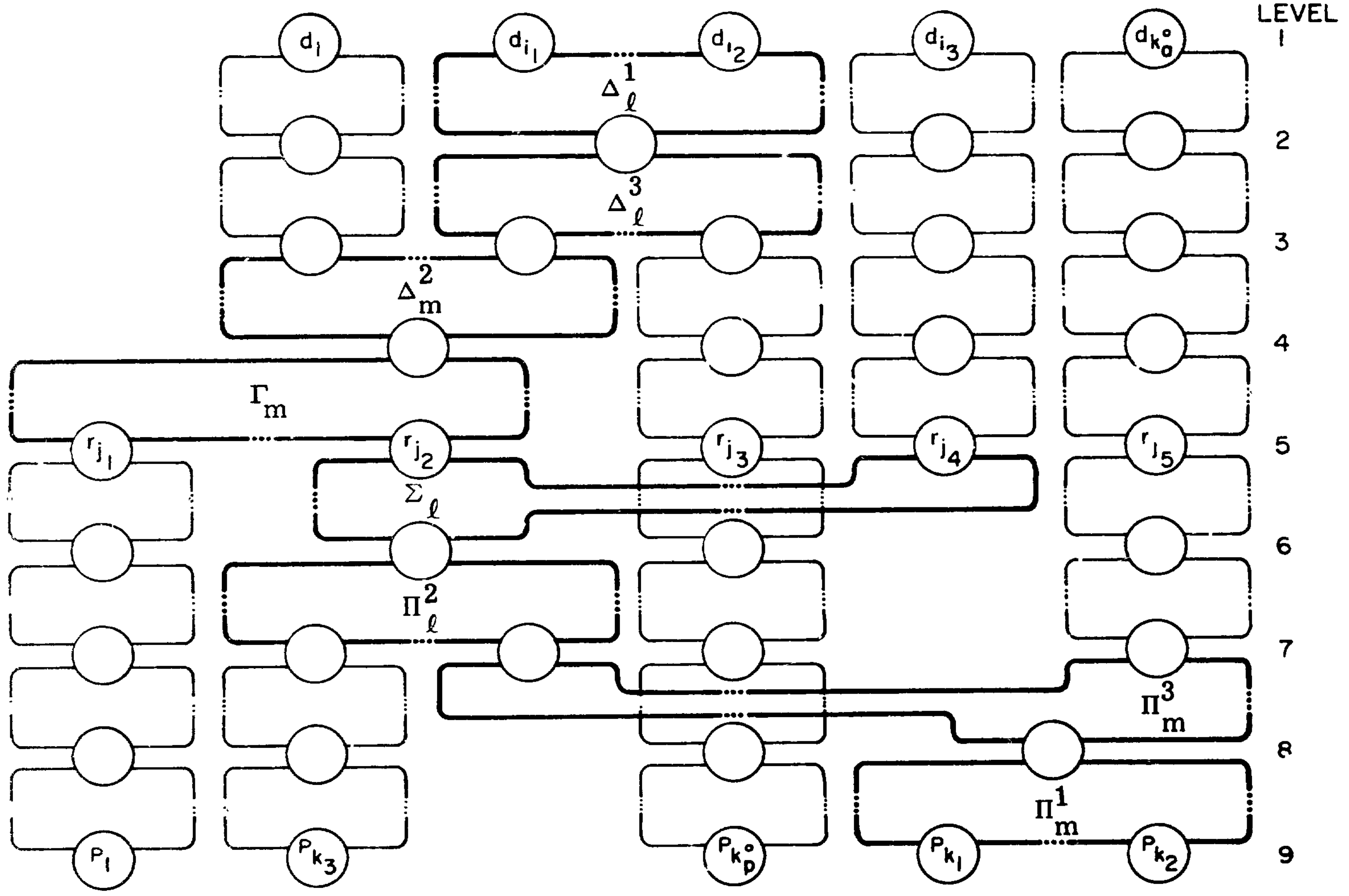


Figure 2-1. Data Structure Model

Figure 2-1 let us point out again the one-to-one correspondence which exists between sets of certain types. For each set $\Delta^1_\ell$ , where $\ell \epsilon \{1, 2, \cdots n_3\}$, there is a corresponding (unique) set $\Delta^3_\ell$. Similarly, for each set $\Delta^2_m$, where $m \epsilon \{1, 2, \cdots n_4\}$, there is a corresponding set $\Gamma_m$; for each set $\Pi^2_\ell$, where $\ell \epsilon \{1, 2, \cdots n_2\}$, there is a corresponding set $\Sigma_\ell$; and for each set $\Pi^1_m$, where $m \epsilon \{1, 2, \cdots, n_1\}$, there is a corresponding set $\Pi^3_m$.

In the structure of Figure 2-1 each node at level 2 (which is a set node) represents a pair of sets $(\Delta^1_\ell, \Delta^3_\ell)$ for $\ell \epsilon \{1, 2, \cdots, n_3\}$. Similarly, each node at level 4 represents a pair of sets $(\Delta^2_m, \Gamma_m)$ for $m \epsilon \{1, 2, \cdots, n_4\}$; each node at level 6 represents a pair of sets $(\Sigma_\ell, \Pi^2_\ell)$ for $\ell \epsilon \{1, 2, \cdots, n_2\}$; and each node at level 8 represents a pair of sets $(\Pi^3_m, \Pi^1_m)$ for $m \epsilon \{1, 2, \cdots, n_1\}$. For each of these set pairs the ring above the corresponding node (i.e., the connective nodes in that ring) indicates the composition of the first set in the pair, and the ring below that node indicates the composition of the second set in the pair.

Each node at level 1 represents some distinct data item $d_i \epsilon \Delta$ and each node at level 9 represents some distinct data item $p_k \epsilon \Pi$, with the result that the set $\Delta$ is represented by the entire collection of nodes at level 1 and the set $\Pi$ is represented by the entire collection of nodes at level 9.

For a given set $\Delta^1_\ell$, which is represented by some node at level 2, the elements contained therein are represented by those nodes at level 1 which are contained within the ring associated with the given node at level 2.

Similarly, for a given set $\Pi^1_m$ which is represented by some node at level 8, the elements contained therein are represented by those nodes at level 9 which are contained within the ring associated with the given node at level 8.

Next consider some set $\Delta^2_m$, which is represented by a node at level 4. The elements of $\Delta^2_m$ are represented by those nodes at level 1 which are associated with the nodes at level 2 which have nodes at level 3 in common with the ring associated with the node (at level 4) representing $\Delta^2_m$. Stated somewhat differently, the sets $\Delta^1_\ell$ which are subsets of $\Delta^2_m$ (and which collectively contain the elements of $\Delta^2_m$) are represented by those nodes at level 2 which share (in their rings that pass through level 3) nodes which are members of the ring that passes through level 3 and is associated with the node which represents $\Delta^2_m$.

We note that any two rings which have at least one node in common have in fact <u>exactly</u> one node in common.

Each set $\Pi^2_\ell$, which is represented by some node at level 6, is treated at levels 6, 7, 8 and 9 in a manner analogous to the sets $\Delta^2_m$.

Consider next some set $\Delta^3_\ell$, which is represented by a node at level 2. The elements of $\Delta^3_\ell$ are represented by nodes at level 4 (which represent the sets $\Delta_m^2$) having rings passing through nodes at level 3 which are members of the ring associated with the node representing $\Delta_\ell^3$.

Again, the analogous situation applies for each set $\Pi_m^3$ at levels 6, 7, and 8.

The only difference between the situations described above and that for the sets $\Gamma_m$ and $\Sigma_\ell$ is that the connective nodes at level 5 represent relation symbols in addition to performing their connectivity functions. For example, given some set $\Gamma_m$, which is represented by a node at level 4, each element of this set is represented by a node at level 5 (representing a relation symbol) and a node at level 6 (representing a target set $\Pi_\ell^2$) which node has a ring passing through the relation symbol node at level 5.

It follows also that each relation $r_j \epsilon P$ (or rather the ordered pairs $(d_i, p_k)$ which form the corresponding solution set $R_j$) is represented by those pairs of nodes at level 1 and 9 which are associated with the nodes at level 5 representing the relation symbol $r_j$. (Note that, in general, there may be more than one node at level 5 which represents the same relation symbol - in different relation instances, of course.)

## 2.2.2 Implications of the Model

Clearly, there is more information by the collection of sets in our data structure model than we have explicitly discussed in our initial description of the schematic of Figure 2-1.

For example, the set $\Delta_{\ell}^{3}$, where $\ell\epsilon\{1,2,\cdots,n_3\}$, indicates which sets $\Delta_{m}^{2}$ contain the set $\Delta_{\ell}^{1}$ as a subset. However, since for every set $\Delta_{m}^{2}$, where $m\epsilon\{1,2,\cdots,n_4\}$, there exists a corresponding set $\Gamma_m$, the set $\Delta_{\ell}^{3}$ also indicates all sets $\Gamma_m$ which are associated with the set $\Delta_{\ell}^{1}$. Hence, the set $\Delta_{\ell}^{3}$ indicates (indirectly) all relation symbol/target set pairs which are associated with the sources in the set $\Delta_{\ell}^{1}$.

In a similar manner the set $\Pi_{m}^{3}$, where $m\epsilon\ \{1,2,\cdots n_1\}$, indicates (indirectly) all source / relation symbol pairs which are associated with the targets of the set $\Pi_{m}^{1}$.

We can, of course, continue with this line of reasoning to determine which targets are associated with a particular source/relation symbol pair, or which sources are associated with a particular relation symbol/ target pair, and so forth. In short, given the sets defined for our data structure model, we can determine any fact concerning the inter-relationships of the elements of a given collection of elementary data items.

### 2.2.3 Uniqueness Considerations

In defining the sets of our data structure model, we have implicitly made a very important decision. If we examine the union of all sets $\Gamma_m$ associated with a given source $d_i$ (as given by the set $\Delta_{\ell}^{3}$ associated with the set $\Delta_{\ell}^{1}$ of which the given source is an element), we find that in each of the relation symbol/target set pairs the relation symbol is unique. In fact, there is a one-to-one correspondence between the elements of this union and the elements of $P_i$. To be more specific, if we let

$$\Gamma_i^o = \{(r_j \Pi_\ell^2) \mid (r_j \Pi_\ell^2) \in \Gamma,\ d_i\ r_j\ \Pi_\ell^2 \in \Omega,\ d_i \in \Delta\}$$

or alternatively

$$\Gamma_i^o = \bigcup_{m \in M_i^o} \Gamma_m$$

where $M_i^o = \{m \mid d_i \in \Delta_m^2 \in \Lambda^*\}$, then the following conditions will hold:

(1) $(r_j \Pi_\ell^2) \in \Gamma_i^o \Longleftrightarrow r_j \in P_i$ for $i \in \{1, 2, \cdots, k_a^o\}$

(2) $r_{j_1} \neq r_{j_2}$ for $(r_{j_1} \Pi_{\ell_1}^2), (r_{j_2} \Pi_{\ell_2}^2) \in \Gamma_i^o$ and $j_1 \neq j_2$

On the other hand, if we examine the union of all sets $\Sigma_\ell$ associated with a given target $p_k$, we find that in each of the source set*/relation symbol pairs the relation symbol is <u>not</u> necessarily unique. That is if we let

$$\Theta_k = \{(\Delta_m^2\ r_j) \mid \Delta_m^2\ r_j\ p_k \in \Omega,\ p_k \in \Pi\}$$

then it is not necessarily the case that $r_{j_1} \neq r_{j_2}$ for $(\Delta_{m_1}^2, r_{j_1})$, $(\Delta_{m_2}^2\ r_{j_2}) \in \Theta_k$ and $j_1 \neq j_2$.

---

* A "source set" is roughly analogous to our notion of a target set. Whereas the sets $\Pi_\ell^2$ are target sets, the sets $\Delta_m^2$ may be considered source sets. Strictly speaking, the elements of a set $\Sigma_\ell$ are source/relation symbol pairs $(d_i r_j)$ and not source set/relation symbol pairs $(\Delta_m^2\ r_j)$.

For ease of reference we will say that this set of conditions implies uniqueness of type 1 .

We may, of course, wish to consider the opposite set of conditions, where the relation symbols in the set of source set/relation symbol pairs associated with a given target $p_k$ are unique, and where the relation symbols in the set of relation symbol/target set pairs associated with a given source $d_i$ are not necessarily unique. We will say that this set of conditions implies uniqueness of type 2. Redefinition of the sets of our data structure model to effect uniqueness of type 2 is straightforward and, therefore, will not be done here.

It should be clear that we cannot require both uniqueness of type 1 and uniqueness of type 2 simultaneously except in very special cases. It should also be clear that it is not desirable to require "non-uniqueness" of both types simultaneously, for in this case our sets and, hence, the model cease to be uniquely defined.

We assume, therefore, that the "problem solver" must specify which type of uniqueness he assumes when he describes the data for his problem in terms of our data structure model.

Since the principles involved are no different whether considering uniqueness of type 1 or of type 2, unless otherwise indicated we will assume uniqueness of type 1 to be in effect for the remainder of the discourse*.

In light of this discussion we can comment further upon the relation symbols represented by the nodes at level 5 of the structure of Figure 2-1.

The nodes designated $r_{j_1}$, $r_{j_2}$, and $r_{j_3}$ (and ostensibly representing the relation symbols $r_{j_1}$, $r_{j_2}$, and $r_{j_3}$) must represent distinct relation symbols since these relation symbols appear in relation symbol/target set pairs which are all elements of the same set $\Gamma^{o}_{i_1}$ corresponding to data item $d_{i_1}$.

It is also true that the relation symbols represented by the nodes designated $r_{j_2}$ and $r_{j_4}$ must be distinct, but for a different reason. In particular, since both relation symbols are associated with the same target set, equality of these relation symbols would imply a single element of the set $\Gamma$ and, hence, a single node in the structure.

On the other hand, the relation symbol represented by the node designated $r_{j_5}$ need not be distinct from either of the relation symbols represented by the nodes designated $r_{j_2}$ and $r_{j_4}$ in spite of the fact that all three relation symbols appear in source set/relation symbol pairs which are elements of the set $\Theta_{k_1}$ corresponding to data item $p_{k_1}$.

---

* Note that the prototype program to which we will refer later in this dissertation has provision for considering both types of uniqueness.

In general, the relation symbol for node $r_{j_5}$ need not be distinct from those for nodes $r_{j_1}$ and $r_{j_3}$, either. Finally, the relation symbol for node $r_{j_3}$ need not be distinct from that for node $r_{j_4}$.

This completes the development of our model for data structure.

## 2.3 Use of the Model

Now that we have a model for data structure, we must specify the manner in which it is to be used. Specific details will be considered in succeeding chapters, but we will provide here certain basic aspects of the use of the model.

The problem solver (i.e., the individual who is confronted with a problem and charged with obtaining its solution) must specify what (or rather, how many) data items and what (again, how many) relations are involved in the solution of his problem. The particular data items and, to a much greater extent, the particular relations which the problem solver chooses to characterize his problem are influenced very strongly by his choice of access processes.

For example, suppose the problem to be solved involves (among others) three classes A, B, and C of data items. Furthermore, suppose there exists a one-to-one correspondence from the elements of each of these classes _onto_ the elements of each of the other two. Finally, suppose that given an element of one of these classes, the corresponding elements of the other two classes are to be determined (i.e., accessed) Assuming that we are given an element $a \in A$ as the intial starting point, we have three choices of procedures for determining the corresponding elements $b \in B$ and $c \in C$:

Procedure 1

(1) Given $a \in A$, determine $b \in B$.

(2) Given $b \in B$, determine $c \in C$.

Procedure 2

(1) Given $a \in A$, determine $c \in C$.

(2) Given $c \in C$, determine $b \in B$.

Procedure 3

(1) Given $a \in A$, determine $b \in B$.

(2) Given $a \in A$, determine $c \in C$.

It follows that we have (at least) three choices of relations which we may define. If procedure 1 is selected to access the various data items, we may define a relation $r_{ab}$ from A to B and a relation $r_{bc}$ from B to C; if procedure 2 is selected, we may define a relation $r_{ac}$ from A to C and a relation $r_{cb}$ from C to B; and finally, if procedure 3 is selected, we may define a relation $r_{ab}$ from A to B and a relation $r_{ac}$ from A to C.

Although this is a very simply example, the concepts which it embodies carry over to much more complex situations.

By examining his data, the problem solver must also determine the cardinalities of the various sets in our data structure model.

Actually, the problem solver need not determine the cardinality of each instance of a set, however, but need only determine the average cardinality of each type of set. For example, rather than determine the actual cardinality of the set $\Delta_{\ell}^{1}$ for each $\ell \in \{1, 2, \cdots, n_3\}$, the problem solver need only determine the average cardinality of the set $\Delta_{\ell}^{1}$ over all $\ell \in \{1, 2, \cdots, n_3\}$. The reason only the average cardinalities are required should become clear in the next chapter.

There is, of course, a very good reason why, from the problem solver's point of view, actual cardinalities are not desirable. Namely, for data structures of even moderate size these cardinalities may be extremely difficult, if not impossible, to determine. In order to do so, one would probably have to construct a structure like that of Figure 2-1 for the entire data structure. On the other hand, as we shall see in Chapter VI, determining average cardinalities for the data structure sets is a relatively painless task.

We might point out, however, that there unfortunately exists no universal technique which may be applied to all problems to determine the average cardinalities of the data structure sets. The process which must be used is very much a function of the particular problem and the availability of information to the problem solver. Nevertheless, an example we consider in Chapter VI should indicate some general guidelines.

It may happen that the mathematical variance of the actual cardinalities for a particular type of set is very large, in which case one might justifiably question the use of an average value. This is particularly true if the cardinalities are characterized by a bimodal distribution for which the average value characterizes neither of the peaks.

Such a case would normally arise where the set of data items may be partitioned into two or more classes among which there is no interaction. For example, suppose that the set of data items consists of the (disjoint) sets A, B, C, D, and E of data items. Suppose further that the elements of these sets are related by the following relations:

$r_{ab}$ from A to B

$r_{da}$ from D to A

$r_{ce}$ from C to E

Clearly, then, there is no "communication" between any element of the sets A, B, and D, and any element of the sets C and E (although elements within the cluster of sets A, B, and D do communicate from one set to another, as do elements within the cluster of sets C and E). We may then partition the set of data items into two noncommunicating classes consisting of the elements of sets A, B, and D and the elements of sets C and E, respectively.

We may treat this situation as consisting of two (or more, as the case may be) distinct problems for which separate data structures may be defined. In a general sense, we may do this even where the boundaries between the classes are not so well defined.

Clearly, the decision to partition a problem into sub-problems lies with the problem solver. For our purposes, we may assume simply that the data structure given is the only one of concern.

Once the average cardinalities for the various data structure sets have been specified by the problem solver, we apply the techniques which are developed in the next three chapters to determine the best storage structure for that data structure.

# Chapter III

# A MODEL OF STORAGE STRUCTURE

In the previous chapter we developed a mathematical model for the intrinsic structure of an arbitrary collection of elementary data items. Our goal in this chapter is the development of a model for all storage structures capable of representing the data structure of that collection of data items.

We desire a model which can assume the form of any of the three basic organizations - sequential, list, and random - or combinations thereof. We also desire a model which is capable of representing the data structure of a collection of data items in varying degrees of detail.

Let us assume that all storage structures to be considered will be resident in a uniform storage medium, each unit of which (such as a byte or word) can be accessed in the same number of time units as any other unit. This type of storage medium is commonly called _random access storage_. The classical example of random access storage is, of course, magnetic-core memory.

We will also make what might be called a storage management disclaimer. We assume that there exists some mechanism for the management of the storage in which our storage structure resides, that is, some overseer which keeps track of those storage locations available for, but not actually occupied by, the structure. This storage supervisor is charged with providing upon request unoccupied storage locations for use by the storage structure and is also charged with reclaiming those storage locations no longer in use by the structure. Although the implementation of these functions is of considerable importance, such considerations do not fall within the scope of our efforts here. Therefore, beyond acknowledging the existence of a storage manager and acknowledging its importance in the implementation of any storage structure, we shall ignore it.

## 3.1 Initial Storage Structure Model

Examine our data structure model as it appears in Figure 2-1. As the first step in the development of a storage structure model, let us develop a single storage structure capable of representing all the detail of the data structure of this model.

Let each node of the DSM (the Data Structure Model of Figure 2-1) be represented by a block of contiguous storage units (bytes, words, etc.) and let each ring of the DSM be represented by a ring of (list) pointers where the unique block of the ring (i.e., the block which represents a node at level 2,4,6, or 8 of the DSM) acts as the head of the ring.

For every pointer thus appearing in a ring let us include a pointer in the opposite direction, resulting in a ring linking all nodes in the direction opposite to the first ring. We will distinguish these two types of rings by saying the first consists of forward pointers and the second consists of reverse pointers (although the distinction as to which ring contains forward pointers and which reverse pointers is purely academic). At times we may also refer simply to "pointers" (other than in the generic sense) in which case we will mean forward pointers.

Let us also include in every ring a pointer from each element block (as distinct from the head) to the head. This pointer we will call a head pointer.

At this point the blocks of our storage structure consist only of pointer fields. We will, of course, have to include information about the data items. Let this information be placed in a number of data item description blocks, one for every elementary data item. Then let each block which represents some data item (i.e., each block which corresponds to a node at level 1 or level 9 of the DSM) contain a pointer to the corresponding data item description block. Each description block (short for data item description block) will then be pointed to by at least one, but no more than two, description block indicators.

In each block representing a node at level 5 of the DSM we will include a field to contain the name (some appropriate code) of the relation symbol which the node represents. Finally, through each block containing the name of a given relation symbol we will pass a ring joining all blocks containing the name of that particular relation symbol. Clearly, the number of such rings will be equal to the cardinality of P. In fact, there will be a one-to-one correspondence between these rings and the relations in P. Each of these rings, which we designate relation rings, will have a head which is distinct from the blocks representing nodes at level 5 of the DSM.

The resultant storage structure, sans reverse and head pointers, is shown schematically in Figure 3-1. We have assumed for the purposes of illustration that the nodes designated $r_{j_3}$ and $r_{j_4}$ in the

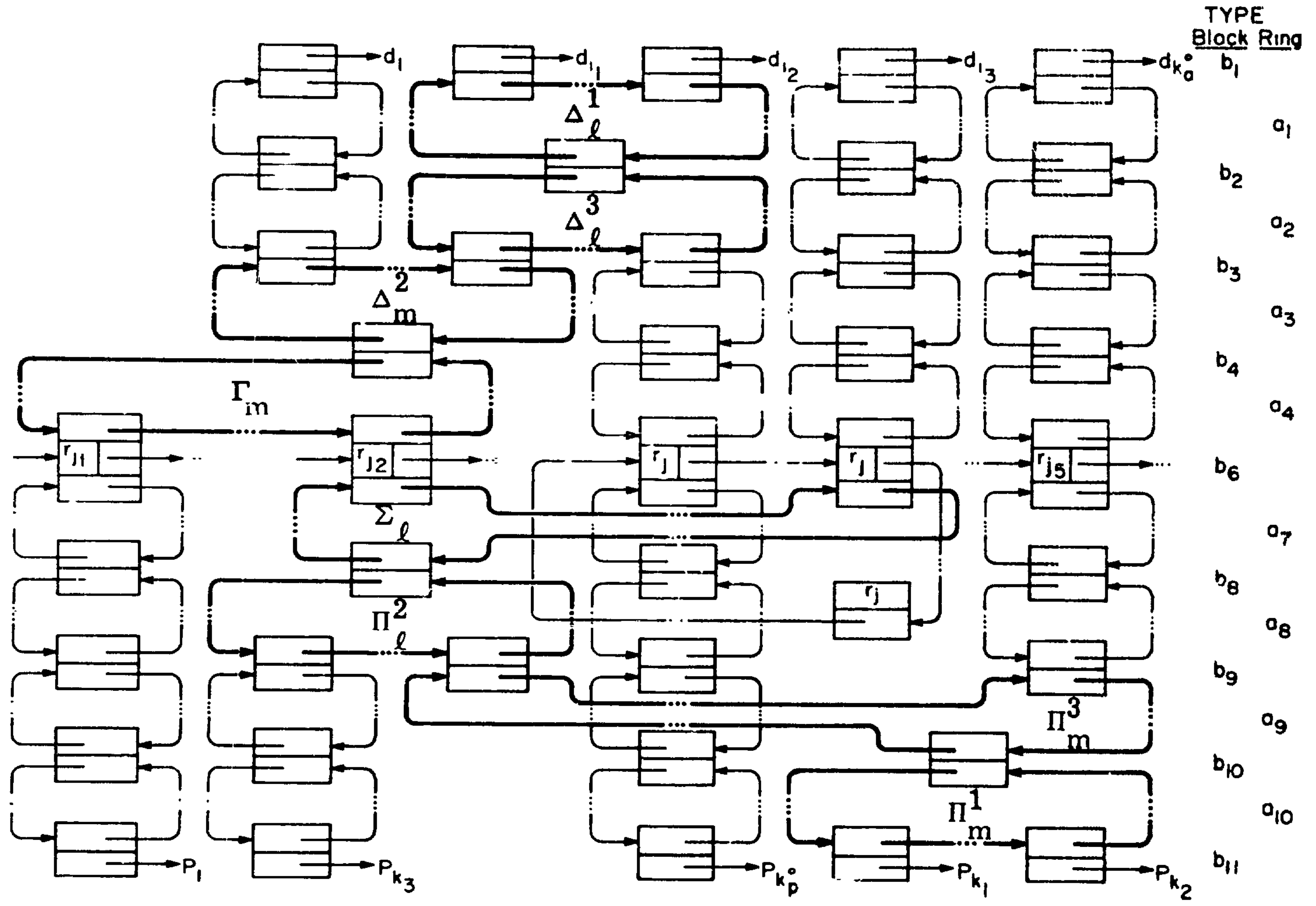


Figure 3-1. Initial Storage Structure Model

DSM represent the same relation symbol $r_j$. We will designate the structure of Figure 3-1 as the Initial Storage Structure Model, or simply, the ISSM. Those pointers emanating from the $b_1$ blocks and the $b_{11}$ blocks and followed by terms of the form $d_i$ and $p_k$, respectively, represent the description block indicators. (The description blocks are not shown.)

## 3.2 Transformations

Instead of the structure described above, we could, of course, have chosen a storage structure in which no head pointers are present, or one in which no reverse pointers are present, or one in which head pointers are present only for certain rings, or one in which some rings are replaced by stacks, and so on.

Clearly, our storage structure model must be flexible enough to allow us to represent variations of this sort. We will achieve this flexibility by applying certain transformations to the ISSM to yield the storage structure of interest.

These transformations, which may be applied individually and in combination to the ISSM, are called

(1) Stacking

(2) Duplication

(3) Elimination

Before considering the effect of each of these transformations upon the ISSM note the symmetry and the repetitive nature of that structure. The blocks designated (by type) $b_2$, $b_4$, $b_8$, and $b_{10}$ in Figure 3-1 each act as the head of two rings, one above and one below the block. On the other hand, the blocks designated $b_3$, $b_6$, and $b_9$ each act as an element in each of two "back-to-back" rings. The blocks designated $b_1$ and $b_{11}$ also act as ring elements but appear in only one ring.

Thus, the ISSM is composed essentially of repetitions of the structure of Figure 3-2. The blocks designated $x_1$ and $x_3$ (specifically, the blocks containing $x_{11}$, $x_{12}$, $x_{31}$, and $x_{32}$) function as the heads of two rings each (only one of which is shown in detail) and the blocks designated $x_2$ (specifically the blocks containing $x_{21}$, $x_{22}$, and $x_{23}$) each function as elements of back-to-back rings.

To simplify further discourse, instead of writing "the blocks designated (by type) $x_1$" we will write "the $x_1$ blocks" and instead of writing "the block containing $x_{11}$" we will write simply "$x_{11}$".

Consider now the effect of each of the three transformations given above upon the structure of Figure 3-2.

The "stacking" transformation causes a given ring to be turned into a stack. For instance, if we apply the stacking transformation to the $z_1$-rings of Figure 3-2, then for each $z_1$-ring we form a stack of $x_2$-blocks (which act as elements of the given $z_1$-ring) upon the corresponding $x_1$-block (which acts as head of the given $z_1$-ring).

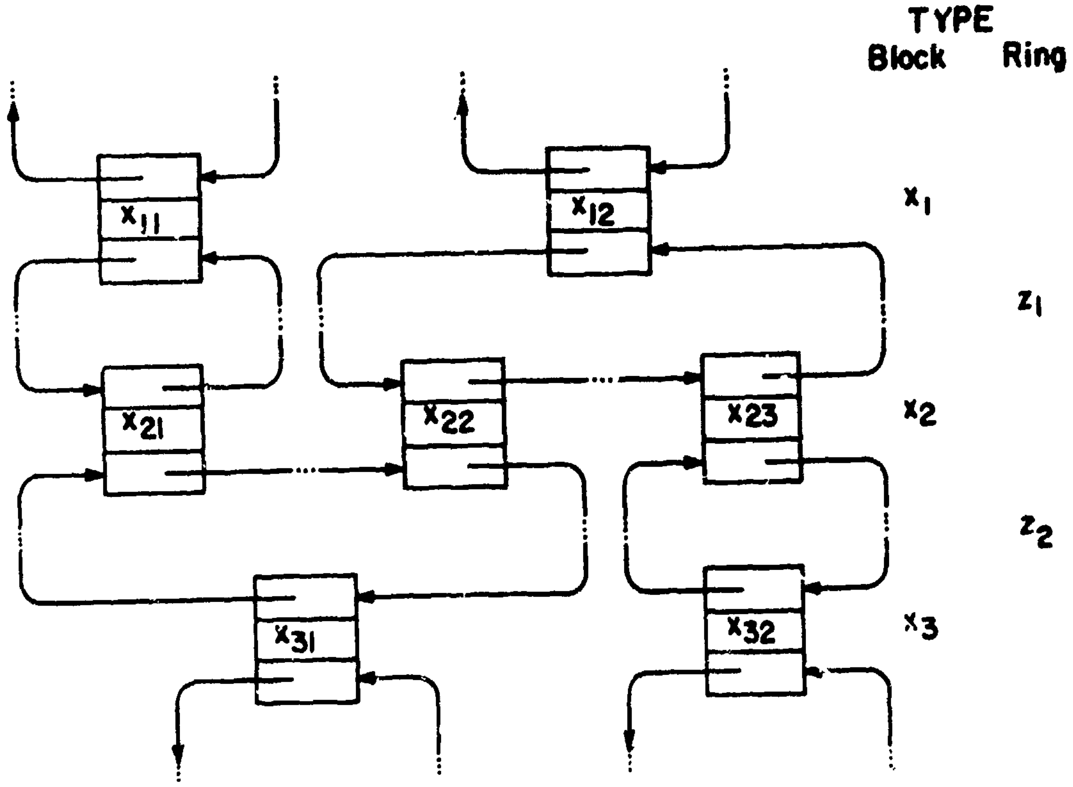


Figure 3-2. A Portion of the Initial Storage Structure Model

Similarly, if we apply the stacking transformation to the $z_2$-rings, then for each $z_2$-ring we form a stack of $x_2$-blocks upon the corre-sponding $x_3$-block. Note that since the $x_2$-blocks are shared by both the $z_1$-rings and the $z_2$-rings, the $x_2$-blocks may be stacked upon either the $x_1$-blocks or the $x_3$-blocks but not both. Figure 3-3 illustrates this transformation as applied to the $z_1$-rings of Figure 3-2.

An alternative to stacking is the "duplication" transformation. Applying this transformation to a ring causes a copy of the head of the ring to be concatenated with each element block of the ring. In addition, for each copy of the head block which is created by this transformation, a copy of the second ring of which the block is head is created. The net result is the removal of a given ring from the structure by duplicating certain other rings. Figure 3-4 illustrates duplication as applied to the $z_1$-rings of Figure 3-2. In order to more clearly illustrate the effect of duplication upon the second ring associated with a duplicated head block, assume that the $x_3$-blocks of Figure 3-2 are duplicated upon the element blocks of the rings below them. The result of this duplication is shown in Figure 3-5. Note that all blocks in the copies of the ring associated with a duplicated block must maintain membership in any other rings of which blocks of the copied ring were members.

Contrary to stacking, duplication may be applied to either the $z_1$-rings or the $z_2$-rings or both simulataneously.

If duplication is applied to both the $z_1$-rings and the $z_2$-rings, the

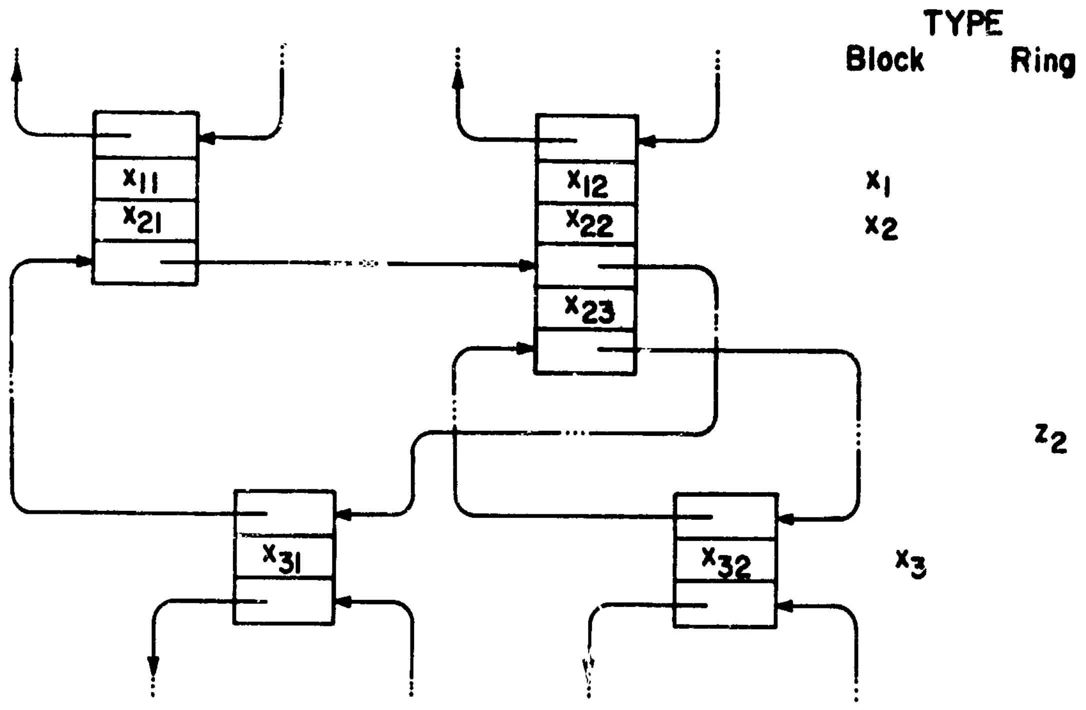


Figure 3-3. Stacking Transformation

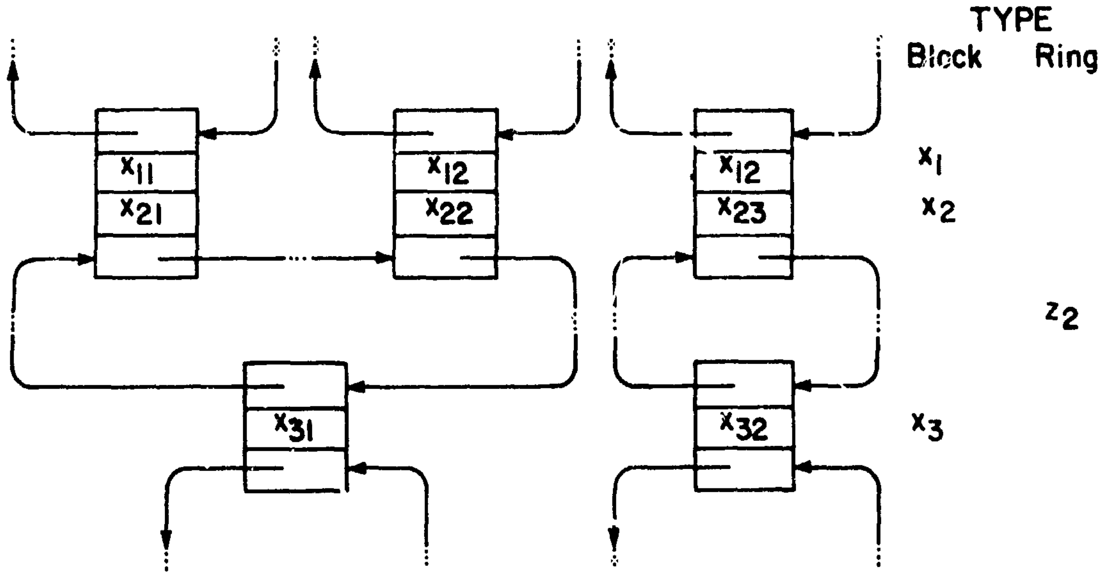


Figure 3-4. Duplication Transformation

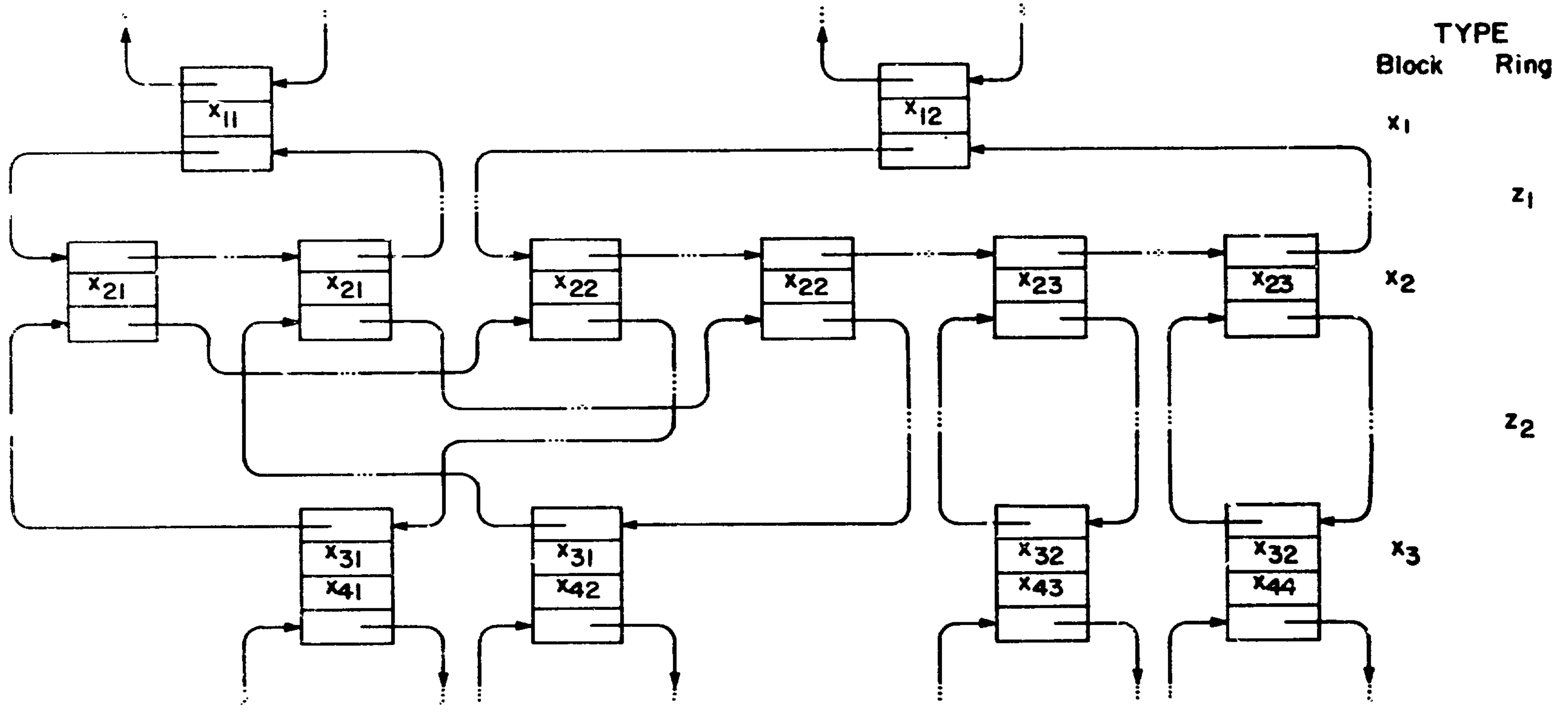


Figure 3-5. Secondary Effects of Duplication Transformation

structure of Figure 3-6 will result. This leads us to our third transformation, "elimination". If the $x_2$-blocks are being used purely for connective purposes (like the $b_3$-blocks and $b_9$-blocks of Figure 3-1), they can be removed, or eliminated, from the structure of Figure 3-6 since in this structure there is a one-to-one correspondence between the $x_1$ -blocks and the $x_3$-blocks which they connect. The resultant structure would appear as in Figure 3-7.

Finally, we can apply stacking to the $z_1$-rings of Figure 3-2 and duplication to the $z_2$-rings or vice-versa which, for the first case, will yield the structure of Figure 3-8.

It should be apparent that stacking, duplication, and elimination can be applied in various combinations simultaneously to the several rings of the ISSM. In order to specify more rigorously just what combinations of these transformations are permissable, we will introduce a number of decision variables which pertain to the various transformations for each of the types of rings within the ISSM. Before doing this, however, we will consider a number of peripheral issues.

## 3.3 Additional Model Characteristics

At first glance it might appear desirable to treat each ring of the ISSM independently from all the others with regard to the transformations which we apply to it. For instance, we may wish to maintain one $a_1$ -ring of the ISSM as a ring, apply the stacking transformation to another $a_1$-ring, and apply the duplication

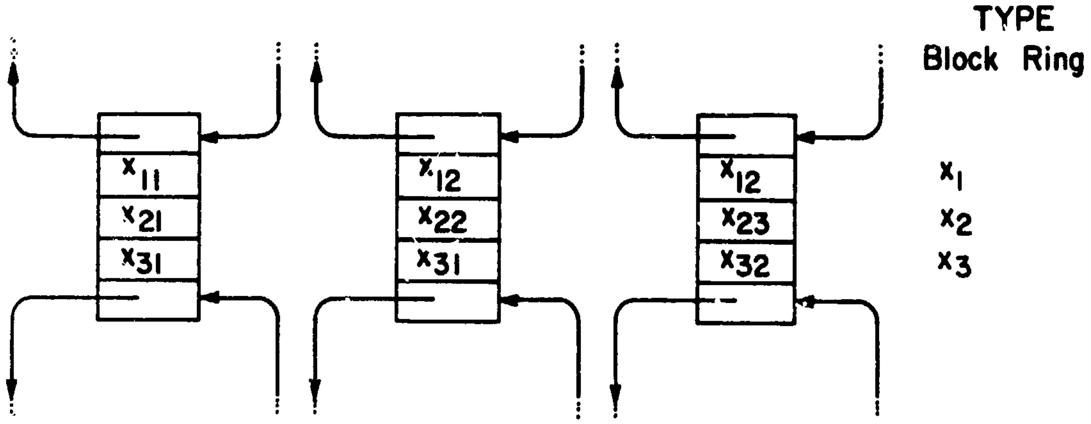


Figure 3-6. Simultaneous Duplication

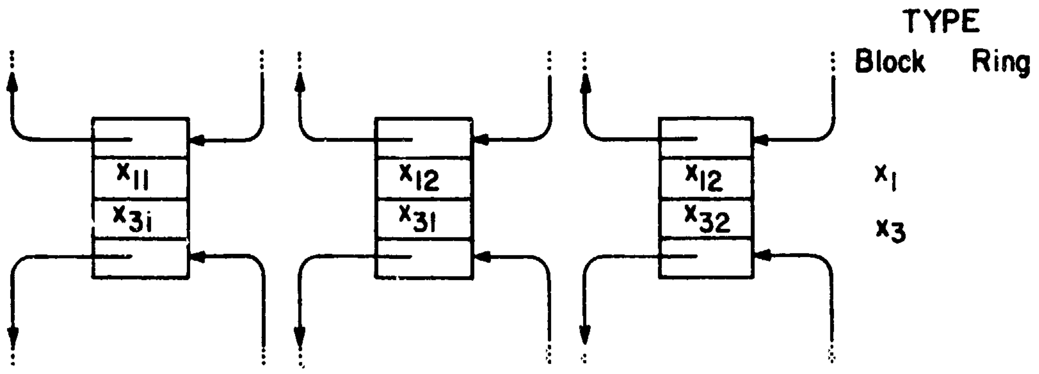


Figure 3-7. Elimination Transformation

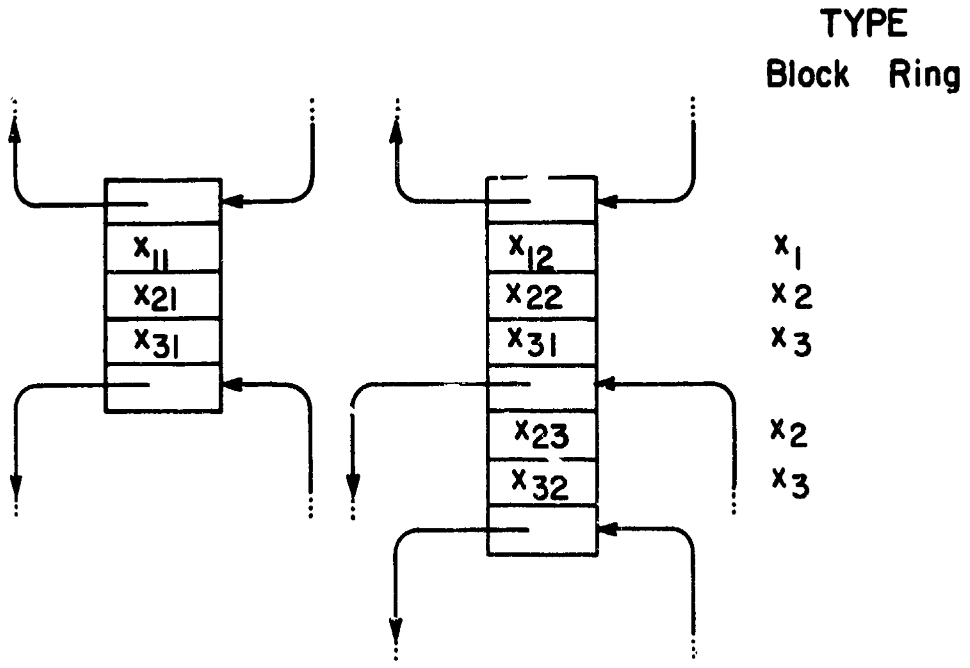


Figure 3-8. Combination of Stacking and Duplication

transformation to still another $a_1$-ring. It quickly becomes apparent, however, that such a posture has two pitfalls. First, it is doubtful that such a structure can be used in a very efficient manner. At each step in the performance of an operation upon the structure one must determine the convention applying to that particular portion of the structure and then choose the proper code to effect the desired step. Second, the number of variables required to specify such a structure via any given model would quickly exceed our ability to consider all possible variations thereof.

Therefore, we will make the assumption that the application of a transformation to any ring of a given type implies the application of that transformation t all rings of the given type. Thus, all rings (and all blocks) of a given type are assumed to be treated uniformly and our model is said to be homogeneous.

As a second issue, notice that applying duplication to the $a_3$-rings of the ISSM or to the $a_8$-rings or to both will in general result in the generation of several copies of each $b_6$-block. (See Figure 3-1.) Clearly, this results in an increase in the number of such blocks in each relation ring. In particular, this results in the generation of two or more $b_6$-blocks to represent a single relation symbol as it appears in the DSM of Figure 2-1. As we shall see later, certain operations which we may want to perform upon the data represented by a particular storage structure may use a given relation ring to search for the occurrence of a particular relation symbol (as used in the DSM and as characterized by its association with one or more relation instances). If we can guarantee that the relation ring contains no more than one $b_6$-block representing the relation symbol sought, then once a $b_6$-block representing the desired relation symbol has been found, the remainder of the $b_6$-blocks in the relation ring may be ignored. On the other hand, if the relation ring may contain more than one $b_6$-block representing the relation symbol sought, then even when a $b_6$-block representing the desired relation symbol has been found, the remainder of the $b_6$-blocks in the relation ring must be examined to check for additional blocks representing the relation symbol.

It may, therefore, be advantageous for the relation rings to maintain their original compositions (and, hence, guarantee that there exists no more than one $b_6$-block representing a given relation symbol) regardless of transformations applied to the rest of the ISSM. For this reason we will now replace each $b_6$-block by the structure of Figure 3-9. We assume, of course, that the rings

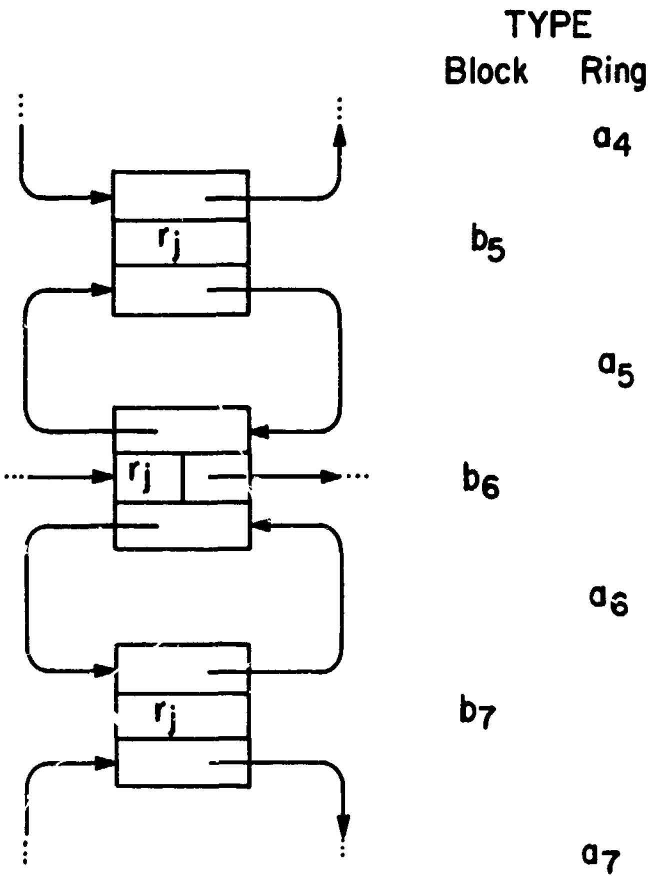


Figure 3-9. Substitute Structure for $b_6$-blocks

therein may (like the rings of the ISSM) contain reverse and head pointers.

The $b_5$-block and the $b_7$-block of Figure 3-9 (each of which contains a field for the name of a relation symbol) replace the $b_6$-block in the $a_4$-ring and in the $a_7$-ring of the ISSM, respectively. In the ISSM upon which no transformations have been made there will then be one $b_5$-block and one $b_7$-block for each (new) $b_6$-block. Applying duplication to the $a_3$-rings and to the $a_8$-rings now results in the generation of copies of $b_5$-blocks and $b_7$-blocks, respectively. Each copy of a $b_5$-block which corresponds to a particular $b_6$-block is put into the $a_5$-ring associated with that $b_6$-block. Similarly, each copy of a $b_7$-block is put into the associated $a_6$-ring.

We see now that the $b_6$-block has assumed a new role in our storage structure model - it now functions as an head (of the $a_5$-ring and the $a_6$-ring) instead of as an element block. Our previous discussions of stacking, duplication, and elimination still apply, however, since the repetitive and symmetric nature of the structure remains unchanged.

Substituting the structure of Figure 3-9 for each $b_6$-block in the ISSM of Figure 3-1 yields the structure of Figure 3-10, which we will henceforth call our Storage Structure Model, or simply SSM.

By making this substitution we have clearly made no changes which would affect the ability of the structure to represent the

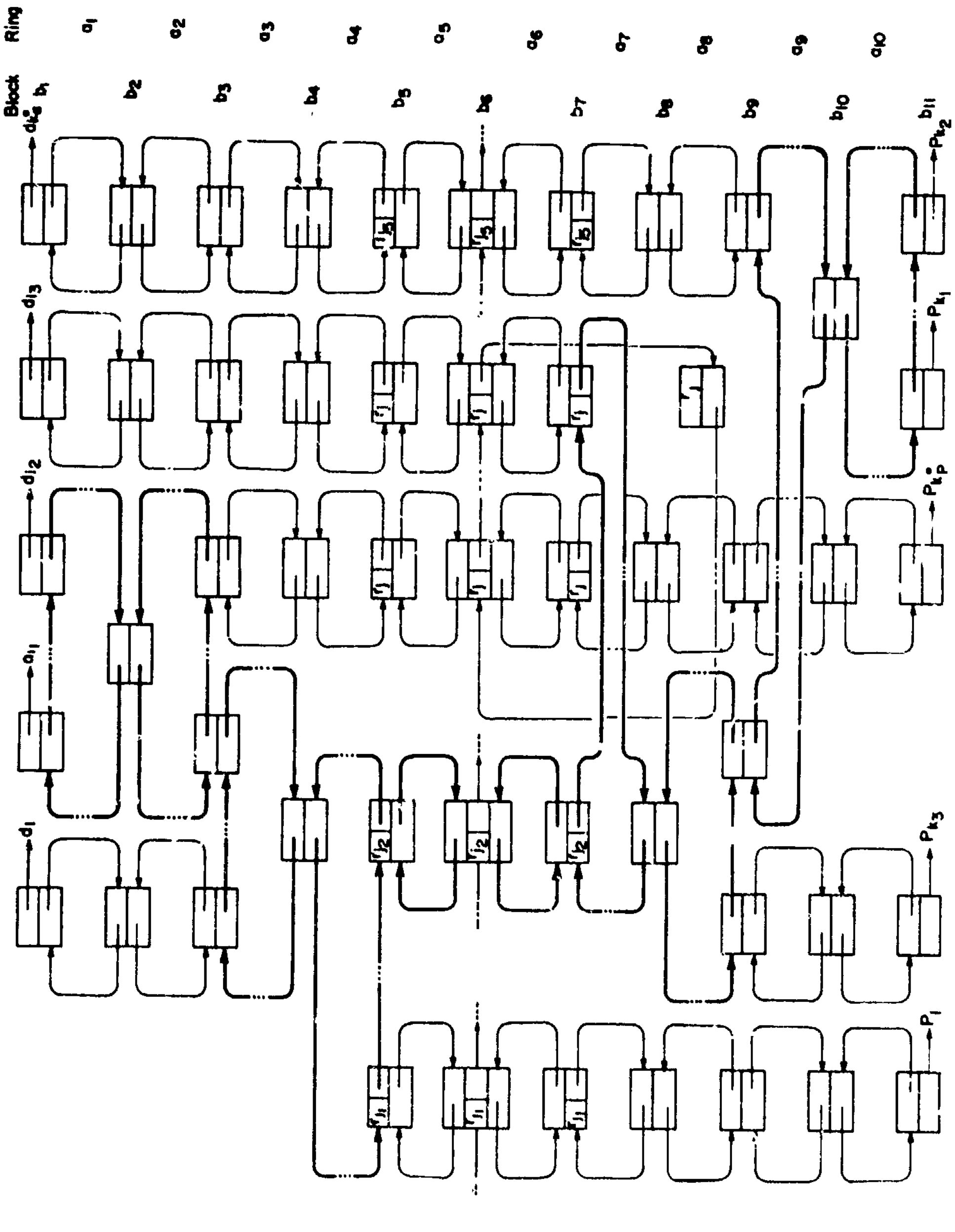


Figure 3-10. Storage Structure Model

intrinsic structure of a collection of data items as modeled by the DSM.

A third issue to consider also involves the effects of duplication. Whenever duplication is applied to the $a_2$-rings or the $a_9$-rings of the SSM, copies of $b_1$-blocks and $b_{11}$-blocks, respectively, are generated. Since certain operations which we may want to perform upon the data represented by a particular storage structure may require the ability to access all copies of a given $b_1$-block or $b_{11}$-block as they represent a given source or target, respectively, we will introduce for each $b_1$-block and each $b_{11}$-block of the SSM a ring to contain all copies of the block generated by duplication. These rings (which are similar to the relation rings for $b_6$-blocks) will be called <u>source rings</u> for $b_1$-blocks and <u>target rings</u> for $b_{11}$-blocks.

As a final issue we introduce the possibility of a block type field for each block of the SSM. In certain cases, especially when stacking and duplication have been applied to several pairs of adjacent rings, it may be advantageous, if not imperative, to have some means for distinguishing the different types of blocks from one another. To accomplish this, we may include within each block a <u>type field</u> which contains some code identifying the type of the block.

Since the SSM contains only eleven distinct types of blocks $(b_1, b_2, \cdots, b_{11})$, a type field need contain at most four bits. If fewer than eleven types are to be distinguished (due to the elimination of certain blocks from the SSM, for instance), an even smaller type field may be used. If some other item in a given type of block does not use the entire field allocated to it (as is frequently true with pointer fields), it may be possible to put the type code in that unused space, thus, eliminating the need for a separate type field. In any event we will provide the option of including a type field in the blocks of any given type.

## 3.4 Decision Variables

Return now to consideration of the decision variables which we will use to describe the SSM and transformations applied thereto.

Let $\phi_i$ be a binary-valued decision variable, the value of which indicates whether ($\phi_i$=1) or not ($\phi_i$=0) forward pointers are present in the $a_i$-rings of the SSM, where $i \epsilon \{1, 2, \cdots, 10\}$. $\phi_i$=0 implies that either the stacking or the duplication transformation has been applied to the $a_i$-rings and that they are no longer really rings at all.

Let $\Delta_i$ be a binary-valued decision variable, the value of which indicates whether the stacking transformation ($\Delta_i$=0) or the duplication transformation ($\Delta_i$=1) has been applied to the $a_i$-rings of the SSM when $\phi_i$=0, where $i \epsilon \{1, 2, \cdots, 10\}$. As a matter of convenience, we will require $\Delta_i$=0 when $\phi_i$=1.

Let $\phi'_i$ be a binary-valued decision variable, the value of which indicates whether ($\phi'_i$=1) or not ($\phi'_i$=0) head pointers are present in the $a_i$-rings of the SSM, where $i \epsilon \{1, 2, \cdots, 10\}$. Note that head pointers may be present even though the $a_i$-rings are not actually rings. For instance, head pointers may be used to advantage when the element blocks of a ring are stacked upon the head.

Let $\phi_i''$ be a binary-valued deciscion variable, the value of which indicates whether ($\phi_i''=1$) or not ($\phi_i''=0$) reverse pointers are present in the $a_i$-rings of the SSM, where $i\epsilon\{1,2,\cdots,10\}$. We will assume that $\phi_i''$ can equal 1 only if $\phi_i=1$. That is, we will assume that reverse pointers are present in the $a_i$-rings only if forward pointers are. What this really means is that if the $a_i$-rings are linked via pointers in one direction only, then these pointers are forward pointers; reverse pointers are present only to provide two-way linking.

Let $\beta_i$ be a binary-valued deciscion variable, the value of which indicates whether ($\beta_i=1$) or not ($\beta_i=0$) the $b_i$-blocks of the SSM are to be eliminated from the structure, where $i\epsilon\{2,3,\cdots,10\}$. (The $b_1$-blocks and the $b_{11}$-blocks are assumed always to be present.) From our discussion of elimination we know that the value of $\beta_i$ is determined by the values of $\Delta_{i-1}$ and $\Delta_i$ where $i\epsilon\{2,3,\cdots,10\}$. In particular, if $\Delta_{i-1}=\Delta_i=1$, then $\beta_i=1$; otherwise $\beta_i=0$. In other words, the $b_i$-blocks are eliminated from the SSM if and only if duplication has been applied to the $a_{i-1}$-rings and to the $a_i$-rings, where $i\epsilon\{2,3,\cdots,10\}$.

Finally, let $\tau_i$ be a binary-valued decision variable, the value of which indicates whether ($\tau_i=1$) or not ($\tau_i=0$) the $b_i$-blocks of the

SSM contain a type field, where $i \in \{1, 2, \cdots, 11\}$.

There are a number of constraints (some of which we have already mentioned) which the above decision variables must satisfy in order to insure a physically realizable structure. Each of these will be considered in turn below.

Constraint 1

$$\phi_i + \Delta_i \leq 1 \text{ for all } i \in \{1, 2, \cdots, 10\}$$

This constraint implies that $\phi_i$ and $\Delta_i$ may not both be equal to 1. That this should be true is obvious: $\phi_i = 1$ implies explicit $a_i$-rings of (forward) pointers and $\Delta_i = 1$ implies the application of duplication to the rings, clearly an impossible situation.

Constraint 2

$$\phi''_i \leq \phi_i \quad \text{for all } i \in \{1, 2, \cdots, 10\}$$

This constraint implies that the $a_i$-rings may not contain reverse pointers unless they contain forward pointers.

Constraint 3

$$\phi'_i + \Delta_i \leq 1 \quad \text{for all } i \in \{1, 2, \cdots, 10\}$$

The purpose of this constraint is to prohibit head pointers from the $a_i$-rings if duplication has been applied to them. Clearly, a head pointer would be to no advantage in a ring to which duplication has been applied since the head is attached directly to each element block.

Constraint 4

$$\beta_i = \Delta_{i-1} \Delta_i \quad \text{for all } i \epsilon \{2,3,\cdots,10\}$$

This constraint implies that the $b_i$-blocks may be (in fact, must be) eliminated from the SSM if and only if duplication has been applied to the rings on both sides of the blocks.

Constraint 5

$$\phi_i + \phi_{i+1} + \Delta_i + \Delta_{i+1} \geq 1 \quad \text{for all } i \epsilon \{2,4,6,8\}$$

The purpose of this constraint is to prohibit the application of stacking to both (types of) rings which share common element blocks.

Upon close examination of the SSM it becomes clear that we must generalize this constraint somewhat to yield the following constraint.

$$\sum_{i=j}^{k} \phi_i + \Delta_j + \Delta_k \geq 1$$

for all $j \epsilon \{2,4,6,8\}$, $k \epsilon \{3,5,7,9\}$, and $k > j$

The implication is that if each type of ring from the $a_j$-rings to the $a_k$-rings inclusive is subjected to either stacking or duplication, then duplication must be applied to either the $a_j$-rings or the $a_k$-rings (or both). If this constraint is not met, we have the (impossible) situation analogous to applying stacking to rings which share common element blocks. (In fact, if $k = j+1$, we have precisely that situation.)

Perhaps a brief example would serve to clarify this point. Assume that $j=2$ and $k=5$ and that duplication has been applied to both the $a_3$-rings and the $a_4$-rings. In this case each combination of related $b_3$-blocks, $b_4$-blocks, and $b_5$-blocks is "fused" into a single block which functions as an element of both an $a_2$-ring and an $a_5$-ring. It follows that stacking may not be applied to both the $a_2$-rings and the $a_5$-rings simultaneously. Since we have assumed that either stacking or duplication has been applied to each, we see that duplication must be applied to either the $a_2$-rings or the $a_5$-rings (or both). We note that if stacking were applied to the $a_3$-rings instead of duplication, then duplication must be applied to the $a_2$-rings. Similarly, if stacking were applied to the $a_4$-rings instead of duplication, then duplication must be applied to the $a_5$-rings.

There are a number of other decisions which we may want to make in characterizing the structure via the

SSM. In particular, instead of assuming that each $b_1$-block and each $b_{11}$-block contains a description block indicator, we may wish to assume that the description blocks are attached directly to the $b_1$-blocks or to the $b_{11}$-blocks or both. Let $\sigma_1$ be a binary-valued decision variable, the value of which indicates whether ($\sigma_1$=1) or not ($\sigma_1$=0) the $b_1$-blocks of the SSM contain description block indicators; if they do not, we assume that the appropriate description block is attached directly to each $b_1$-block. Let $\sigma_2$ be a binary-valued description variable defined similarly for the $b_{11}$-blocks of the SSM.

When we defined the data item description block, we indicated that there is one description block for every data item. In examining transformations which may be applied to the SSM we have seen that it is possible for the transformed structure to contain more than one copy of a $b_1$-block and/or more than one copy of $b_{11}$-block. We wish to reaffirm at this point the assumption that there is indeed exactly one description block for each data item. This means, of course, that if $\sigma_1 = 1$, we must insure that there is exactly one copy of each $b_1$-block in the transformed structure. Similarly, if $\sigma_2=0$, we must insure that there is exactly one copy of each $b_{11}$-block in the transformed structure. We will discuss the enforcement of these two constraints somewhat later in the discourse.

The assumption of one description block for each data item perhaps requires some justification. Possibly, the most compelling reason for making the assumption is that the data item is the smallest, atomic unit of information to be represented and, hence, it should be represented in the structure by some unique, well-defined, closed-form device.

Consider a simple example. Suppose that the value (i.e., the description) of a given data item varies in the course of the solution of a problem. For instance, the data item might correspond to the coordinates of a symbol being displayed upon a computer-driven graphical (CRT) display. If the symbol is being moved across the display, its coordinates will vary with time and must constantly be updated. Since all descriptions of such a data item must be altered each time its value changes, a single description is to be greatly preferred.

A second reason for making the assumption is that when a data item is both a target and a source, the description block acts in a connective capacity, much as the element blocks which are shared by back-to-back rings. Any two such rings share at most one element block between them. Thus, if the analogy is to carry over fully, there should be only one description block for a given data item.

The connective role of the description block brings up another point. If $\sigma_1=0$ and $\sigma_2=0$, it is clear that we may not apply stacking to both the $a_1$-rings and the $a_{10}$-rings. It should also be clear that (except in special cases) whenever $\Delta$ and $\Pi$ are not disjoint, there must be at least one $i\in\{1, 2, \cdots, 10\}$ such that the $a_i$-rings of the SSM are explicit rings (i.e., $\phi_i=1$). This, of course, is to allow the structure to "wrap around". Thus, we have the following constraints.

Constraint 6

$$\sigma_1+\sigma_2+\phi_1+\phi_{10}+\Delta_1+\Delta_{10}\geq 1$$

This constraint disallows stacking of both the $a_1$-rings and the $a_{10}$-rings when $\sigma_1=0$ and $\sigma_2=0$.

Constraint 7

$$\text{If } \sigma_1=0,\ \sigma_2=0,\ \text{and } |\Delta|+|\Pi|>|\Delta^0|,\ \text{then}$$

$$\sum_{i=1}^{10}\phi_i\geq 1$$

This constraint implies the existence of at least one explicit ring whenever $\Delta$ and $\Pi$ are not disjoint. (We note that $|\Delta|+|\Pi| \geq |\Delta^o|$ is always true. Hence, $|\Delta|+|\Pi|=|\Delta^o|$ implies $\Delta$ and $\Pi$ are disjoint.) We can relax this constraint somewhat to allow $\sum_{i=1}^{10} \phi_i=0$ in cases for which $\Delta$ and $\Pi$ are not disjoint provided we can guarantee that $d_{i_1}\ r_{j_1}\ d_{i_2}$ and $d_{i_2}\ r_{j_2}\ d_{i_1}$ are not both true for any $d_{i_1}$ and $d_{i_2} \epsilon \Delta^o$ and any $r_{j_1}$ and $r_{j_2} \epsilon$ P ($d_{i_1}$ and $d_{i_2}$ are not necessarily distinct, nor are $r_{j_1}$ and $r_{j_2}$).

The next set of decisions we may want to make involves the presence of the relation symbol name field in the $b_5$-blocks, the $b_6$-blocks, and the $b_7$-blocks. We may desire to exclude this field from certain of these blocks. Therefore, we introduce the binary-valued decision variables $\rho_1, \rho_2$, and $\rho_3$, where $\rho_1$ indicates whether ($\rho_1=1$) or not ($\rho_1=0$) the relation symbol name field is present in the $b_5$-blocks of the SSM, and $\rho_2$ and $\rho_3$ perform similar functions for the $b_6$-blocks and the $b_7$-blocks, respectively.

Up to this point we have assumed that elimination of the $b_i$-blocks from the SSM always occurs if duplication has been applied to the $a_{i-1}$-rings and to the $a_i$-rings, where $i \epsilon \{2,3,\cdots,10\}$. (See Constraint 4.) We wish now to alter this assumption somewhat so that if the b -blocks contain relation symbol fields, they will not be eliminated from the structure under any conditions. Clearly, this change applies only for $i \epsilon \{5,6,7\}$.) Therefore, we replace Constraint 4 by the following one.

Constraint 4'

For $i \epsilon \{2,3,4,8,9,10\}$

$$\beta_i = \Delta_{i-1}\Delta_i$$

For $i \epsilon \{5,6,7\}$

If $\rho_{i-4} = 1$, $\beta_i = 0$

Otherwise, $\beta_i = \Delta_{i-1}\Delta_i$

This concludes our discussion of the decision variables requir<sup></sup>ed to describe the SSM.

## 3.5 Quantification of Model

Now that we have a formal manner (i.e., the decision variables) for describing the transformations which may be applied to the SSM, we would like to determine the quantitative effects upon the SSM of applying various transformations.

Assume that for each of the sets of the DSM $\Delta_\ell^1$, $\Delta_\ell^3$, $\Delta_m^2$, $\Gamma_m$, $\Sigma_\ell$, $\Pi_\ell^2$, $\Pi_m^3$, and $\Pi_m^1$ (for all possible values of their respective indices) we are given the expected number of elements therein (i.e., the average cardinality of the set). The expected number of elements contained in some set $\Delta_\ell^1$, where $\ell \epsilon \{1, 2, \cdots, n_3\}$, can, for example, be determined from

$$\frac{1}{n_3} \sum_{\ell=1}^{n_3} |\Delta_\ell^1|$$

This means that we know the expected number of element blocks in each type of ring of the SSM before any transformations have been applied. In particular, let $k_i^o$ represent the expected number of element blocks in each of the $a_i$-rings of the SSM (before the application of transformations), where $i \epsilon \{1, 2, \cdots, 10\}$. $k_5^o$ and $k_6^o$ will, of course, both be 1.

Consider now the effect of transformation of the SSM upon the expected number of element blocks in each of the various ring types. Let $k_i$ represent the expected number of element blocks in each of the $a_i$-rings of the SSM after the application of transformations, where $i \epsilon \{1, 2, \cdots, 10\}$. Clearly, the application of stacking to the $a_i$-rings has no effect upon $k_i$ - it remains equal to $k_i^o$. Furthermore, we will assume that elimination has no effect upon $k_i$. Even though the element blocks of the $a_i$-rings may be physically absent, the function which they perform is not altered by this fact. We assume that $k_i$ represents in this case the expected number of "virtual" element blocks. It follows that $k_i$ is affected only by duplication. (We know from our earlier discussions that $k_i$ will indeed be affected by duplication.)

For notational convenience, if $\lambda$ is a binary-valued decision variable, we define $\bar{x}$ as having a value which is the complement of the the value of x. That is, if $x=0$, then $\bar{x}=1$, and if $x=1$, then $\bar{x}=0$.

We may then write expressions governing the various $k_i$ for $i\epsilon\{1,2,\cdots,10\}$ as follows:

$k_1=\Delta_1+\bar{\Delta}_1k_1^o$

$k_2=\Delta_2+\bar{\Delta}_2k_2^o(\bar{\Delta}_4+\Delta_4k_4^o(\bar{\Delta}_6+\Delta_6k_6^o(\bar{\Delta}_8+\Delta_8k_8^o(\bar{\Delta}_{10}+\Delta_{10}k_{10}^o))))$

$k_3=\Delta_3+\bar{\Delta}_3k_3^o(\bar{\Delta}_1+\Delta_1k_1^o)$

$k_4=\Delta_4+\bar{\Delta}_4k_4^o(\bar{\Delta}_6+\Delta_6k_6^o(\bar{\Delta}_8+\Delta_8k_8^o(\bar{\Delta}_{10}+\Delta_{10}k_{10}^o)))$

$k_5=\Delta_5+\bar{\Delta}_5k_5^o(\bar{\Delta}_3+\Delta_3k_3^o(\bar{\Delta}_1+\Delta_1k_1^o))$

$k_6=\Delta_6+\bar{\Delta}_6k_6^o(\bar{\Delta}_8+\Delta_8k_8^o(\bar{\Delta}_{10}+\Delta_{10}k_{10}^o))$

$k_7=\Delta_7+\bar{\Delta}_7k_7^o(\bar{\Delta}_5+\Delta_5k_5^o(\bar{\Delta}_3+\Delta_3k_3^o(\bar{\Delta}_1+\Delta_1k_1^o)))$

$k_8=\Delta_8+\bar{\Delta}_8k_8^o(\bar{\Delta}_{10}+\Delta_{10}k_{10}^o)$

$k_9=\Delta_9+\bar{\Delta}_9k_9^o(\bar{\Delta}_7+\Delta_7k_7^o(\bar{\Delta}_5+\Delta_5k_5^o(\bar{\Delta}_3+\Delta_3k_3^o(\bar{\Delta}_1+\Delta_1k_1^o))))$

$k_{10}=\Delta_{10}+\bar{\Delta}_{10}k_{10}^o$

We can easily verify the validity of these expressions. Consider first $k_1$. If $\Delta_1=0$, each $a_1$-ring of the SSM will be either an explicit ring ($\phi_1=1$) of $k_1^o$ $b_1$-blocks or a stack ($\phi_1=0$) of $k_1^o$ $b_1$-blocks upon a $b_2$-block. ($k_1^o$ is, of course, the expected - not actual - number of blocks in each of these rings or stacks.) If on the other hand

$\Delta_1=1$, each $a_1$-ring of the transformed SSM will consist simply of a $b_1$-block and a (copy of a) $b_2$-block, concatenated. Clearly, applying the duplication transformation - or not applying it - to any other type of ring in the SSM can have no effect upon the number of elements in the $a_1$-rings. (Applying duplication to the $a_2$-rings, for instance, will affect the number of $a_1$-rings but not the number of elements contained in each.) The expression for $k_1$ follows directly.

Consider next $k_3$. Assume for the moment that $\Delta_1=0$. Then $k_3$ will behave exactly as $k_1$: if $\Delta_3=0$, each $a_3$-ring of the SSM will be either an explicit ring ($\phi_3=1$) of $k_3^o$ $b_3$-blocks or a stack ($\phi_3=0$) of $k_3^o$ $b_3$-blocks upon a $b_4$-block; and if $\Delta_3=1$, each $a_3$-ring will consist of a $b_3$-block concatenated with a $b_4$-block. On the other hand if $\Delta_1=1$, each $b_2$-block will be replaced by $k_1^o b_3$-blocks, one for each time the $b_2$-block of an $a_1$-ring is duplicated upon a $b_1$-block. In this case, if $\Delta_3=0$, each $a_3$-ring or stack will contain $k_1^o k_3^o$ $b_3$-blocks. If $\Delta_3=1$, of course, each $a_3$-ring will still consist of a single $b_3$-block concatenated with a $b_4$-block. Again, applying the duplication transformation to any rings other than the $a_1$-rings or the $a_3$-rings can have no effect upon the number of elements in the $a_3$-rings. The expression for $k_3$ follows directly.

Using arguments similar to those above, we can easily justify the expressions given for the remaining $k_i$.

In addition to the number of element blocks in each $a_i$-ring (i.e., the number of element blocks associated directly with the head of each $a_i$-ring), we may desire to know the number of element blocks not actually in each $a_i$-ring but associated indirectly with the head of that ring. For instance, we may want to know the number of $b_5$-blocks associated with a particular $b_2$-block. Since there are $k_2$ $a_4$-rings associated with each $b_2$-block (from the fact that the $a_2$-ring of which the $b_2$-block is head contains $k_2$ $b_3$-blocks and there is one $a_4$-ring associated with each of these $b_3$-blocks) and since there are $k_4$ $b_5$-blocks in each $a_4$-ring, we determine that there are $k_2k_4$ $b_5$-blocks associated (indirectly) with each $b_2$-block.

For the moment let the notation $b_i/b_j$ designate the number of $b_i$-blocks associated (either directly or indirectly) with each $b_j$-block. Let us then define a 5 by 6 matrix K which has as its elements those quantities indicated by Table 3-1. Using arguments similar to that used above to determine $b_5/b_2=k_2k_4$, we may determine values for the remaining elements of K. These values are given by Table 3-2. We will denote the element in the j-th column of the i-th row of K by $K_{ij}$ where $i\epsilon \{1,2,\cdots,5\}$ and $j\epsilon \{1,2,\cdots,6\}$.

It might be well at this point to consider a number of other

| | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| 1 | $b_1/b_2$ | $b_3/b_2$ | $b_5/b_2$ | $b_7/b_2$ | $b_9/b_2$ | $b_{11}/b_2$ |
| 2 | $b_1/b_4$ | $b_3/b_4$ | $b_5/b_4$ | $b_7/b_4$ | $b_9/b_4$ | $b_{11}/b_4$ |
| 3 | $b_1/b_6$ | $b_3/b_6$ | $b_5/b_6$ | $b_7/b_6$ | $b_9/b_6$ | $b_{11}/b_6$ |
| 4 | $b_1/b_8$ | $b_3/b_8$ | $b_5/b_8$ | $b_7/b_8$ | $b_9/b_8$ | $b_{11}/b_8$ |
| 5 | $b_1/b_{10}$ | $b_3/b_{10}$ | $b_5/b_{10}$ | $b_7/b_{10}$ | $b_9/b_{10}$ | $b_{11}/b_{10}$ |

Table 3-1. Definition of Elements of Matrix K.

|  | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| 1 | $k_1$ | $k_2$ | $k_2k_4$ | $k_2k_4k_6$ | $k_2k_4k_6k_8$ | $k_2k_4k_6k_8k_{10}$ |
| 2 | $k_1k_3$ | $k_3$ | $k_4$ | $k_4k_6$ | $k_4k_6k_8$ | $k_4k_6k_8k_{10}$ |
| 3 | $k_1k_3k_5$ | $k_3k_5$ | $k_5$ | $k_6$ | $k_6k_8$ | $k_6k_8k_{10}$ |
| 4 | $k_1k_3k_5k_7$ | $k_3k_5k_7$ | $k_5k_7$ | $k_7$ | $k_8$ | $k_8k_{10}$ |
| 5 | $k_1k_3k_5k_7k_9$ | $k_3k_5k_7k_9$ | $k_5k_7k_9$ | $k_7k_9$ | $k_9$ | $k_{10}$ |

Table 3-2. Values of Elements of Matrix K.

quantities which are similar in many respects to those just considered and which will prove useful somewhat later.

Let $m_a$ represent the number of copies of each $b_1$-block generated by transformation of the SSM. Before any transformations are applied to the SSM, $m_a=1$, of course. Using an argument similar to that used in obtaining expressions for the various $k_i$, we obtain the following expression for $m_a$.

$$m_a=\overline{\Delta}_2+\Delta_2k_2^O(\overline{\Delta}_4+\Delta_4k_4^O(\overline{\Delta}_6+\Delta_6k_6^O(\overline{\Delta}_8+\Delta_8k_8^O(\overline{\Delta}_{10}+\Delta_{10}k_{10}^O))))$$

Similarly, let $m_p$ represent the number of copies of each $b_{11}$-block generated by transformation of the SSM.

$$m_p=\overline{\Delta}_9+\Delta_9k_9^O(\overline{\Delta}_7+\Delta_7k_7^O(\overline{\Delta}_5+\Delta_5k_5^O(\overline{\Delta}_3+\Delta_3k_3^O(\overline{\Delta}_1+\Delta_1k_1^O))))$$

Let $m_{r_1}$ represent the number of copies of a given $b_5$-block associated with the (copies of the) $b_1$-block representing a given source.

$$m_{r_1}=\overline{\Delta}_6+\Delta_6k_6^O(\overline{\Delta}_8+\Delta_8k_8^O(\overline{\Delta}_{10}+\Delta_{10}k_{10}^O))$$

Finally, let $m_{r_2}$ represent the number of copies of a given $b_7$-block associated with the (copies of the) $b_{11}$-block representing a given target.

$$m_{r_2}=\bar{\Delta}_5+\Delta_5 k^o_5(\bar{\Delta}_3+\Delta_3 k^o_3(\bar{\Delta}_1+\Delta_1 k^o_1))$$

We indicated earlier that when $\sigma_1=0$, we must insure that there is no more than one copy of each $b_1$-block in the SSM, and when $\sigma_2=0$, we must insure that there is no more than one copy of each $b_{11}$-block. We now have the means to enforce these two constraints. If $\sigma_1=0$, we must restrict $\Delta_2, \Delta_4, \Delta_6, \Delta_8$, and $\Delta_{10}$ to values which guarantee $m_a=1$. Similarly, if $\sigma_2=0$, we must restrict $\Delta_1, \Delta_3, \Delta_5$, $\Delta_7$, and $\Delta_9$ to values which guarantee $m_p=1$. In particular, $\bar{\sigma}_1\ m_a \leq 1$ and $\bar{\sigma}_2\ m_p \leq 1$.

Our discussions so far have been concerned mainly with the number of blocks of one type which are associated with a given block of some other type. We will, however, also have need for the total number of blocks of a given type.

Let $m_i$ represent the (average) total number of $b_i$-blocks in the SSM, where $i\epsilon\{1, 2, \cdots, 11\}$.

There are $k^o_a$ sources and $m_a$ $b_1$-blocks representing each source in the SSM. Therefore, $m_1=m_a k^o_a$. Since there are $k_1$ $b_1$-blocks associated with each $b_2$-block, we know that $m_2=m_1/k_1$. Associated with each $b_2$-block are $k_2$ $b_3$-blocks. Hence, $m_3=k_2 m_2$. Clearly, $m_4=m_3/k_3$, and so on. Carrying out all indicated multiplications and divisions will result in the expressions given in Table 3-3.

$m_1 = m_a k_a^o$

$m_2 = \frac{m_1}{k_1} = \frac{m_a k_a^o}{k_1}$

$m_3 = k_2 m_2 = \frac{k_2 m_a k_a^o}{k_1}$

$m_4 = \frac{m_3}{k_3} = \frac{k_2 m_a k_a^o}{k_1 k_3}$

$m_5 = k_4 m_4 = \frac{k_2 k_4 m_a k_a^o}{k_1 k_3}$

$m_6 = \frac{m_5}{k_5} = \frac{k_2 k_4 m_a k_a^o}{k_1 k_3 k_5}$

$m_7 = k_6 m_6 = \frac{k_2 k_4 k_6 m_a k_a^o}{k_1 k_3 k_5}$

$m_8 = \frac{m_7}{k_7} = \frac{k_2 k_4 k_6 m_a k_a^o}{k_1 k_3 k_5 k_7}$

$m_9 = k_8 m_8 = \frac{k_2 k_4 k_6 k_8 m_a k_a^o}{k_1 k_3 k_5 k_7}$

$m_{10} = \frac{m_9}{k_9} = \frac{k_2 k_4 k_6 k_8 m_a k_a^o}{k_1 k_3 k_5 k_7 k_9}$

$m_{11} = k_{10} m_{10} = \frac{k_2 k_4 k_6 k_8 k_{10} m_a k_a^o}{k_1 k_3 k_5 k_7 k_9}$

Table 3-3. Numbers of Blocks, by Type, in the SSM

We also know, of course, that $m_{11}=m_p k^o_p$. Hence, the following equation must always hold.

$$k_1 k_3 k_5 k_7 k_9 \; m_p k^o_p = k_2 k_4 k_6 k_8 k_{10} \; m_a k^o_a$$

In particular, it must hold for the SSM before the application of any transformations:

$$k^o_1 k^o_3 k^o_5 k^o_7 k^o_9 k^o_p = k^o_2 k^o_4 k^o_6 k^o_8 k^o_{10} k^o_a$$

Recalling that $k^o_5 = 1$ and $k^o_6 = 1$, we finally get the equation

$$k^o_1 k^o_3 k^o_7 k^o_9 k^o_p = k^o_2 k^o_4 k^o_8 k^o_{10} k^o_a$$

We will later find this relationship to be of some use to us.

Let us derive another expression which is, at the least, of some academic interest. Assume that no transformations have been applied to the SSM. The number of $b_6$-blocks associated with a given $b_1$-block is easily determined to be $k_2^o k_4^o$. This number, however, also corresponds to the number of relation symbols associated with a given source (i.e., $|P_i|$ where $d_i \epsilon \Delta$ is the data item represented by the given $b_1$-block). The number of $b_{11}$-blocks associated with a given $b_6$-block is $k_8^o k_{10}^o$ (recalling that $k_6^o = 1$), and this number corresponds to the number of targets associated with a given source/relation symbol pair. Therefore, the number of relation symbol/target pairs associated with a given source must be $k_2^o k_4^o k_8^o k_{10}^o$. We know in addition that $\Delta$ contains $k_a^o$ sources. It follows then that the total number of relation instances $(d_i\ r_j\ p_k)$ to be represented by the SSM (and described by the DSM) is $k_2^o k_4^o k_3^o k_{10}^o k_a^o$. Using the result obtained above, we may say that the total number of relation instances is also given by $k_1^o k_3^o k_7^o k_9^o k_p^o$.

Continuing along these lines, we know that the number of targets associated with a given source/relation symbol pair cannot exceed the number of targets in $\Pi$. Therefore,

$$k_8^o k_{10}^o \leq k_p^o$$

Simi'arly, the number of sources associated with a given relation symbol/target set pair cannot exceed the number of sources in Δ. Therefore,

$$k_1^o k_3^o \leq k_a^o$$

Also, the number of relation symbols associated with a given source cannot exceed the number of relations in P. As a result,

$$k_2^o k_4^o \leq k_r^o$$

The same thing is true with regard to the number of (distinct) relation symbols associated with a given target. Recall, however, that a given relation symbol may be associated with the same target more than once. If we assume that the expected number of times the same relation symbol is associated with a given target is represented by $m_{r_p}$, then we can say

$$\frac{k_7^o k_9^o}{m_{r_p}} \leq k_r^o$$

Since each relation in P must appear in the SSM at least once, the number of $b_6$-blocks must be at least as great as $|P|$. Hence,

$$\frac{k_2^o k_4^o k_a^o}{k_1^o k_3^o} \geq k_r^o$$

This last result suggests another. Let $m_r$ represent the expected number of $b_3$-blocks which contain the same relation symbol. Since the SSM contains $m_6$ $b_6$-blocks and since there are $k_r^o$ distinct relation symbols, it follows that

$$m_r = \frac{m_6}{k_r^o} = \frac{k_2 k_4 m_a k_a^o}{k_1 k_3 k_5 k_r^o}$$

which becomes

$$m_r = \frac{k_2^o k_4^o k_a^o}{k_1^o k_3^o k_r^o}$$

if no transformations have been applied to theSSM.

## 3.6 Flexibility of Model

The reader has undoubtedly noticed that despite our stated desire to design a storage structure model capable of assuming the form of any of the three basic storage organizations, we have made no mention of the random organization. Be assured, we have not forgotten.

Our reason for ignoring the random organization to this point is simply that the random organization is basically the same as the list organization, the principal difference being that in a list organization a given item contains an explicit pointer to another item, whereas in a random organization the given item contains a key which is used to determine in some way a pointer to the other item. The result is that in getting from one item to another the random organization will generally be slower than (or at least different in time from) the list organization (since one must look in a dictionary, do some calculation, etc. in order to determine the pointer). Later we will see that this difference can easily be reflected by assuming different amounts of time for following a list pointer and for following a random "pointer". Since in general a key is required for each random "pointer", the number of fields (although possibly different in size) will be the same as for a comparable list organization. However, in comparing the storage requirements of the two organizations one must be careful to include the storage required of a dictionary, if that particular method of implementation for the random organization is used, since it will not be included explicitly in our model.

Thus, our model does indeed encompass the random organization without extension or modification.

Let us examine now the flexibility of our model. At the very least we would like some assurance that the SSM is capable of representing such basic structures as a simple stack and a simple linear list.

Presupposing a structure such as a stack or linear list assumes, of course, that the intrinsic structure of the data to be represented is linear in nature. This means that each relation to be represented is one-to-one. That is, a given data item is related to exactly one other data item by some given relation and vice versa. If this is true, it follows that $k_i^o = 1$ for all $i \epsilon \{1, 2, \cdots, 10\}$. The SSM will then appear as in Figure 3-11.

Suppose that we assign the following values to our various decision variables:

$$\phi_i = \phi'_i = \phi''_i = 0 \quad \text{for all } i \epsilon \{1, 2, \cdots, 10\}$$
$$\Delta_i = 1 \quad \text{for all } i \epsilon \{1, 2, \cdots, 10\}$$
$$\tau_i = 0 \quad \text{for all } i \epsilon \{1, 2, \cdots, 11\}$$
$$\sigma_1 = 0, \ \sigma_2 = 1$$
$$\rho_1 = \rho_3 = 0, \ \rho_2 = 1$$

It follows from Constraint 4' that $\beta_6 = 0$ and $\beta_i = 1$ for all $i \epsilon \{2, 3, \cdots 10\}$ and $i \not\leq 6$. The SSM will now appear as in Figure 3-12, where the double-walled field of each block corresponds to a description block.

If there is no occasion to access a relation instance via the relation symbol, we may choose to eliminate the relation rings from the structure. (We assume that the relation rings are always present, but if our analysis should show that they are never used, we may obviously eliminate them from the structure.) We may make

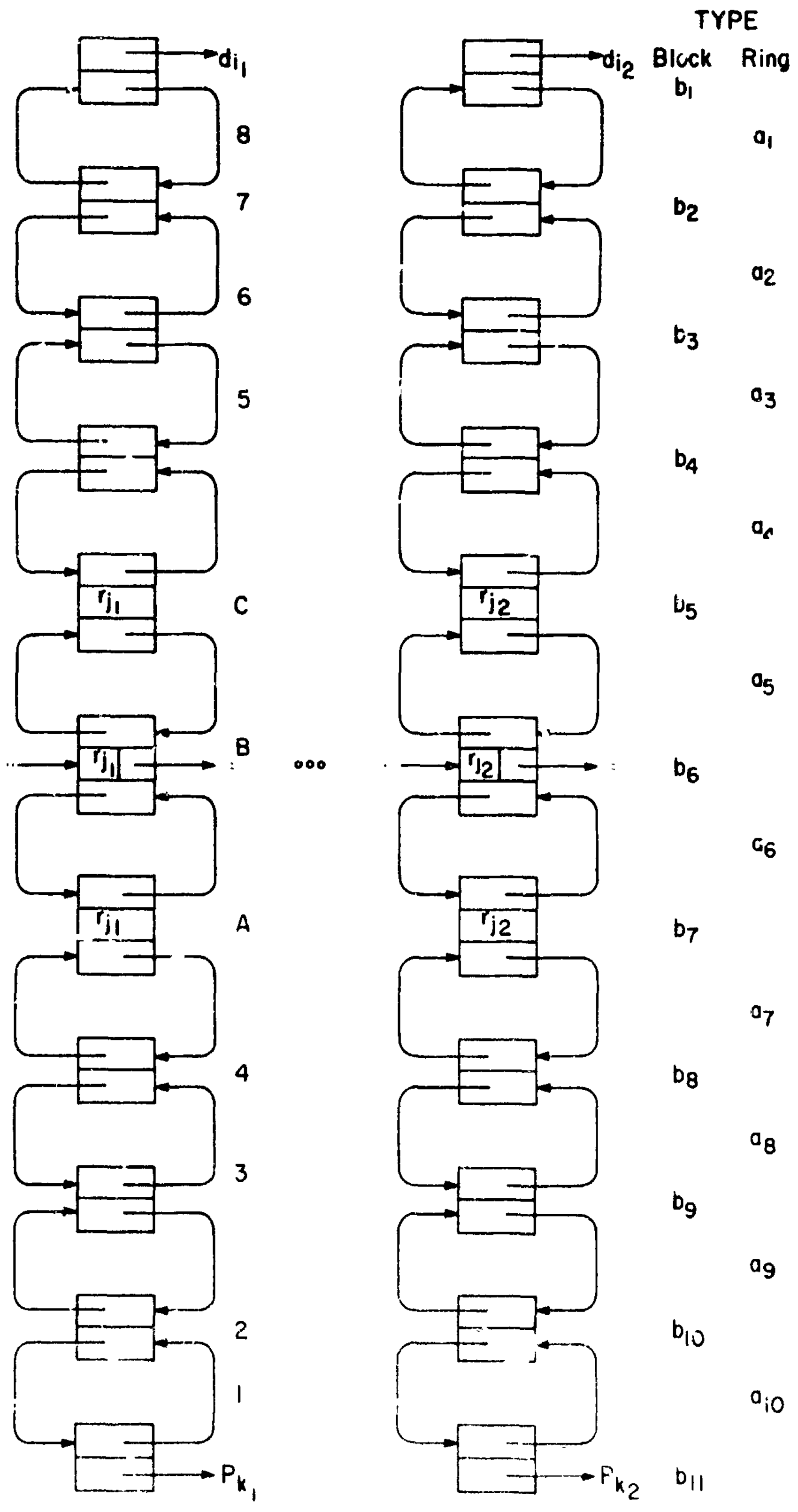


Figure 3-11. Storage Structure Model for all $k_i^O=1$

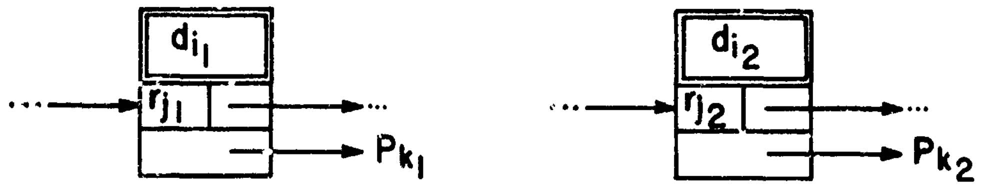


Figure 3-12. Transformed Storage Structure Model - 1

similar decisions about the source and target rings (not shown in the figure). Furthermore, if P contains but a single relation or if we are not interested in the individal relation symbols per se, we may set $\rho_2 = 0$ and eliminate the relation symbol name field from each block. Assuming that we choose all of these options, the SSM will appear as in Figure 3-13 (where we have assumed for illustration that $\Delta$ and $\Pi$ are not disjoint and that any two data items are related by at most one relation).

Clearly, the structure shown in Figure 3-13 is a simple linear list. By setting $\sigma_2 = 0$ it can be transformed into a simple stack. Thus, we have satisfied our first requirement, being capable of representing the very basic storage structures.

As another example to illustrate the flexibility of the SSM and its ability to represent in varying degrees of detail the structure of the DSM, suppose we are given some collection of data items for which the untransformed SSM appears exactly as shown in Figure 3-10 (i.e., with all dotted lines made solid). Furthermore, suppose that the only information which we wish to "share" is that concerning target sets. That is, we want to represent explicitly only the composition of each target set and the source/relation symbol pairs associated with that target set. Clearly, all other "shared" items will be implicit in the resultant structure but will require

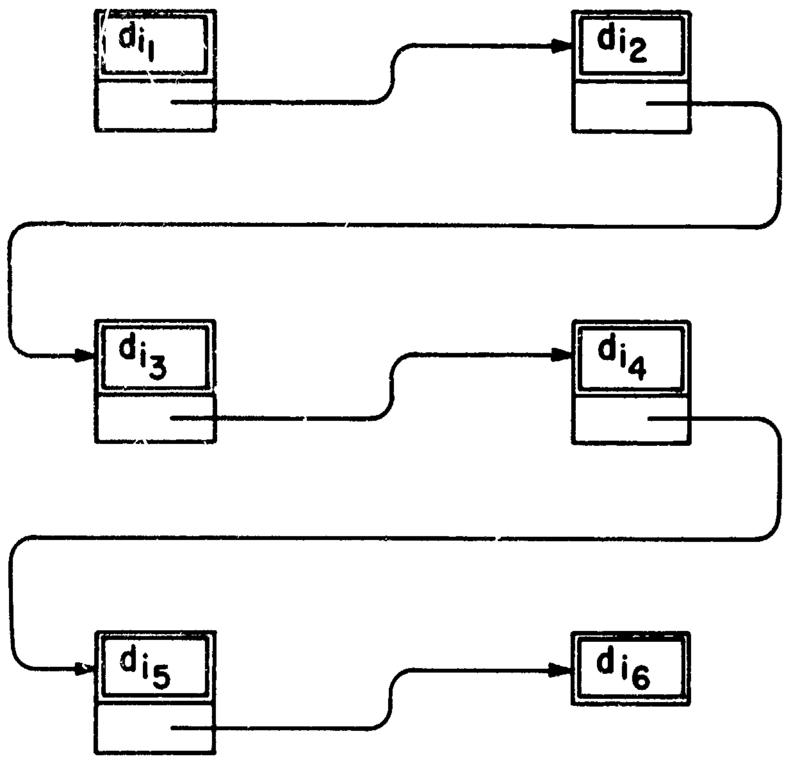


Figure 3-13. Transformed Storage Structure Model - 2

interrogation of the structure to determine.

Let us assign the following values to our various decision variables:

$$\phi_1 = \phi_2 = \phi_3 = \phi_4 = \phi_5 = \phi_6 = 0, \ \phi_7 = \phi_8 = 1, \ \phi_9 = \phi_{10} = 0$$

$$\phi_i' = \phi_i'' = 0 \qquad \text{for all } i \in \{1, 2, \cdots, 10\}$$

$$\Delta_1 = \Delta_2 = \Delta_3 = \Delta_4 = \Delta_5 = \Delta_6 = 1, \ \Delta_7 = \Delta_8 = 0, \ \Delta_9 = \Delta_{10} = 1$$

$$\tau_i = 0 \qquad \text{for all } i \in \{1, 2, \cdots, 11\}$$

$$\sigma_1 = 0, \ \sigma_2 = 1$$

$$\rho_1 = \rho_3 = 0, \ \rho_2 = 1$$

By Constraint 4' $\beta_2 = \beta_3 = \beta_4 = \beta_5 = 1$, $\beta_6 = \beta_7 = \beta_8 = \beta_9 = 0$, $\beta_{10} = 1$. The SSM will then appear as in Figure 3-14. Each $a_8$-ring in this structure represents a target set and each $a_7$-ring represents the set of all source/relation symbol pairs which are associated with a given target set.

It is evident that using the SSM we can represent as much or as little of the detail of the DSM as we wish.

As a somewhat different example, let us consider how the software-simulated associative memory designed by Feldman (and mentioned in Chapter I) could be specified within the context of the SSM.

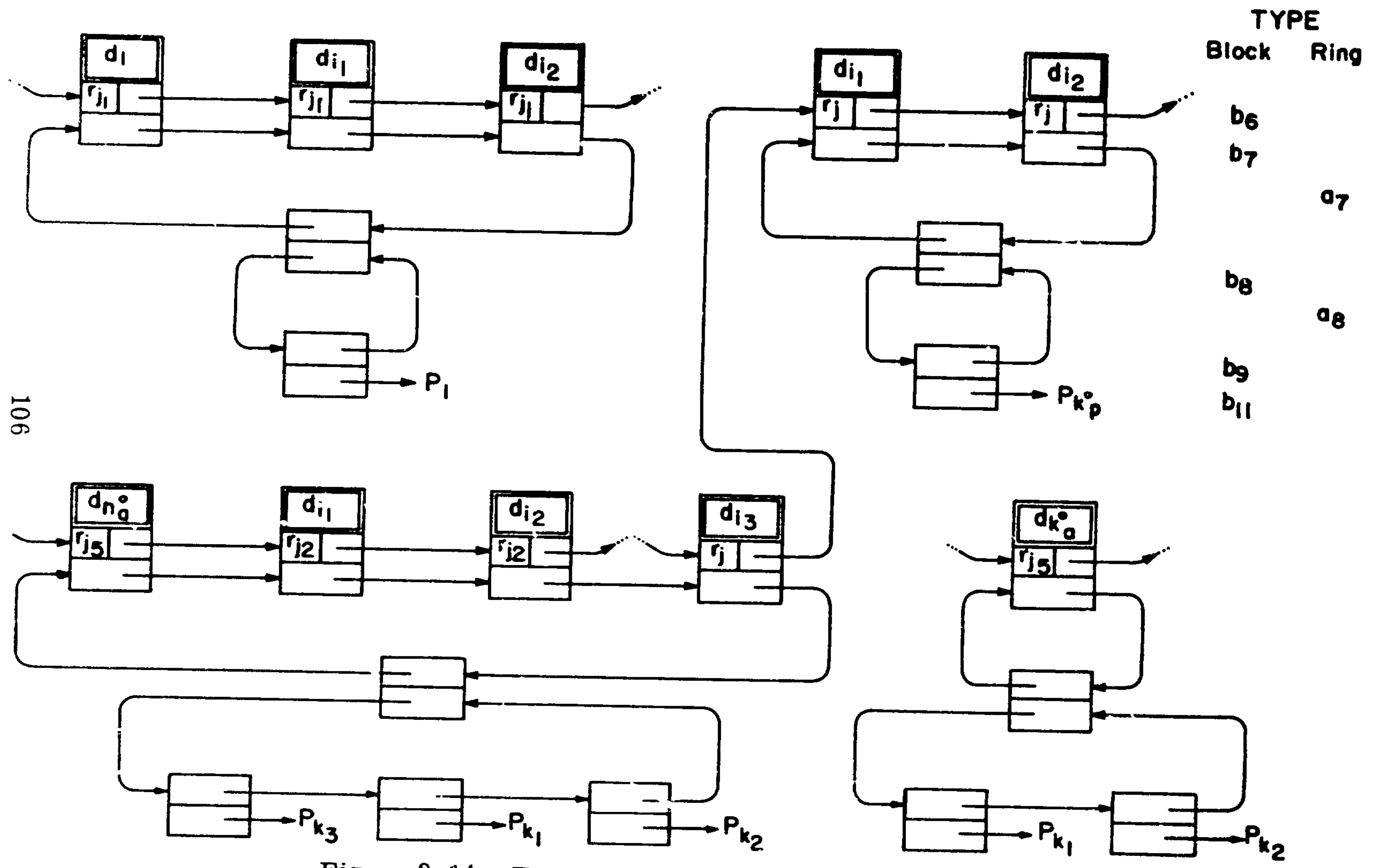


Figure 3-14. Transformed Storage Structure Model - 3

In Feldman's implementation each cell of the "memory" contains five fields as shown in Figure 3-15, three information fields $F_1$, $F_2$, and $F_3$ and two link fields, $L_1$ and $L_2$. Recall that the address of a given cell is determined by hashing the contents of fields $F_1$ and $F_2$.

The $L_1$ field of a cell is used to associate additional cells (of the same form as that shown in Figure 3-15) with the cell accessed by hashing $F_1$ and $F_2$ in order to handle the multiplicity problem (i.e., the situation in which more than one value for the field $F_3$ is associated with a given combination of values for fields $F_1$ and $F_2$) and the overlap problem (i.e., the situation in which different $F_1/F_2$ value pairs hash to the same address). In particular, the $L_1$ field contains a pointer to one of the cells in a ring of cells containing the value combinations of all triples ($F_1$,$F_2$,$F_3$) for which the values of $F_1$ and $F_2$ hash to the given address.

The $L_2$ field, on the other hand, is used to link in rings all cells having the same values for the field $F_3$. These rings make it possible to determine the values of fields $F_1$ or $F_2$ or both given a value for field $F_3$.

Suppose, for instance, we are given the following triples of values for the fields $F_1$, $F_2$, and $F_3$ (in that order):

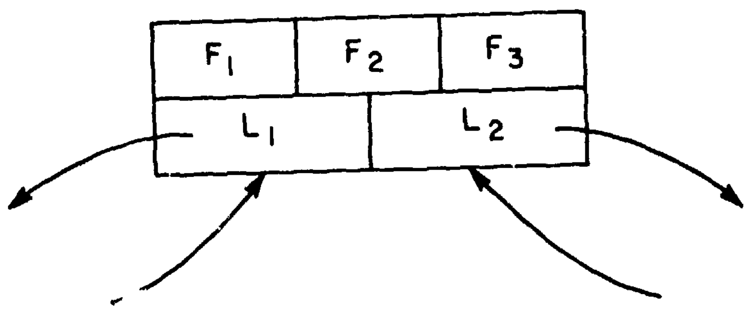


Figure 3-15. Software-Simulated Associative Memory Cell

$$(a_1, b_1, c_1)$$
$$(a_1, b_1, c_2)$$
$$(a_2, b_2, c_3)$$
$$(a_3, b_3, c_3)$$

where the combination of $a_1$ and $b_1$ hashes to the same address as the combination of $a_2$ and $b_2$. The resulting structure would appear as shown in Figure 3-16, where the dashed pointers represent addresses determined by hashing the indicated pairs of values.

Let us define our data items and the corresponding data structure for this case as follows:

(1) Let the set $\Xi$ consist of all (ordered) triples $(a_{i_1}, b_{i_2}, c_{i_3})$ of values for the fields $F_1, F_2$, and $F_3$ to be represented by the storage structure.

(2) $\Delta = \{(a_{i_1}, b_{i_2}) \mid (a_{i_1}, b_{i_2}, c_{i_3}) \in \Xi$ for some $c_{i_3}\}$

(3) $\Pi = \{c_{i_3} \mid (a_{i_1}, b_{i_2}, c_{i_3}) \in \Xi$ for some $a_{i_1}$ and $b_{i_2}\}$

(4) Let P contain a single relation r from $\Delta$ to $\Pi$ with the solution set R such that

$$((a_{i_1}, b_{i_2}), c_{i_3}) \in R \Longleftrightarrow (a_{i_1}, b_{i_2}, c_{i_3}) \in \Xi$$

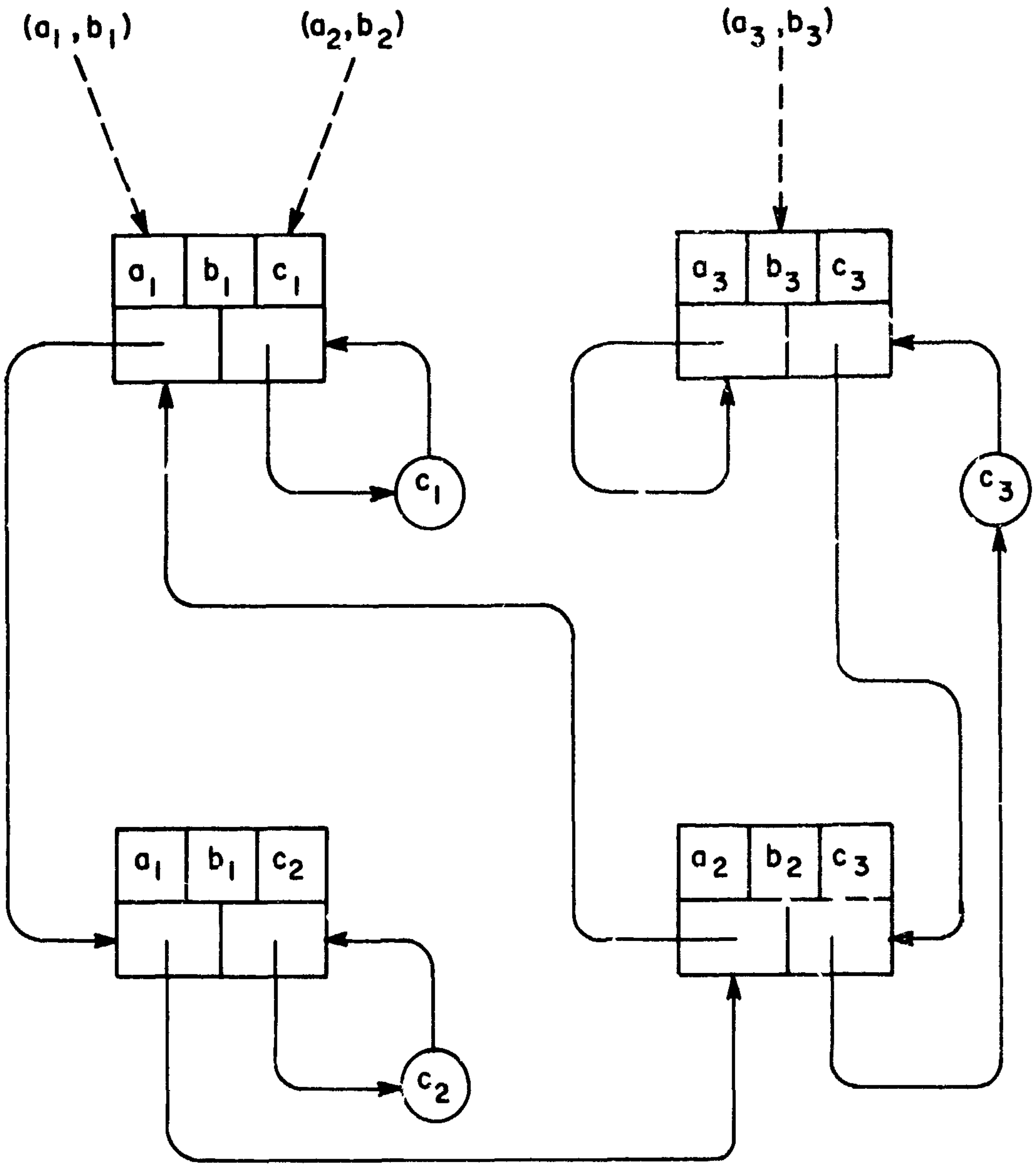


Figure 3-16. Software-Simulated Associative Memory Structure

Since in his storage structure Feldman permits - in fact, if conditions warrant, requires - the duplication of data item descriptions, we shall relax (for this case) our earlier assumption that data item description blocks may not be duplicated.*

Let us assign the following values to our decision variables:

| | |
|---|---|
| $\phi_i = \phi'_i = \phi''_i = 0$ | for all $i \epsilon \{1, 2, \cdots, 10\}$ |
| $\Delta_i = 1$ | for all $i \epsilon \{1, 2, \cdots, 10\}$ |
| $\tau_i = 0$ | for all $i \epsilon \{1, 2, \cdots, 11\}$ |
| $\sigma_1 = \sigma_2 = 0$ | |
| $\rho_1 = \rho_2 = \rho_3 = 0$ | |

It follows from Constraint 4' that $\rho_i = 1$ for all $i \epsilon \{2, 3, \cdots, 10\}$.

These decision variable values result simply in the stack structure described earlier, except in this case the sets $\Delta$ and $\Pi$ are disjoint and the relation considered is not one-to-one. Since we have allowed description blocks to be duplicated, the storage structure consists of isolated blocks of storage each of which contains the values of one of the triples in $\Xi$. Passing through each of these blocks is a source ring, a relation ring, and a target ring.

* To be completely general, we could include in our model a decision variable which indicates whether or not multiple data item descriptions are to be allowed. Although this can very easily be done, we choose not to do so.

If we assume that (1) instead of a source ring for each distinct source in $\Delta$, we have a source ring for each set of sources which hash to a common address, (2) instead of a separate head for each source ring, we use that $b_1$-block which has the address to which all sources in the corresponding set hash, and (3) the relation ring (of which there is only one) is discarded, then the resultant structure is identically equal to Feldman's. The source rings correspond to Feldman's rings of $L_1$ links and the target rings correspond to his rings of $L_2$ links.

We can easily justify the two assumptions concerning the source rings. First, combining into one source ring all sources which hash to a common address is reasonable (if not mandatory), since these sources are indistinguishable unless a different method for determining the location of the head of a source ring is used. Second, using one of the elements (although a special element) of a source ring as head instead of creating a special, distinct head does not alter the structure or utility of that ring. In fact, this may always be preferable, since it reduces the size of the ring.

As a final somewhat more complex example, let us consider how Childs' [11] storage representation of his so-called Set-Theoretic Data Structure (STDS)* could be specified within the context of the SSM.

* Note that Childs' use of the term data structure corresponds roughly to our definition of the term storage structure.

Childs defines a Set-Theoretic Data Structure as a storage representation of sets and set operations such that given any family N of sets and any collection S of set operations, an STDS is any storage representation which is isomorphic to N with S. In particular, an STDS - shown schematically in Figure 3-17 - is composed of five structurally independent parts:

(1) a collection S of set operations

(2) a set B of data item names

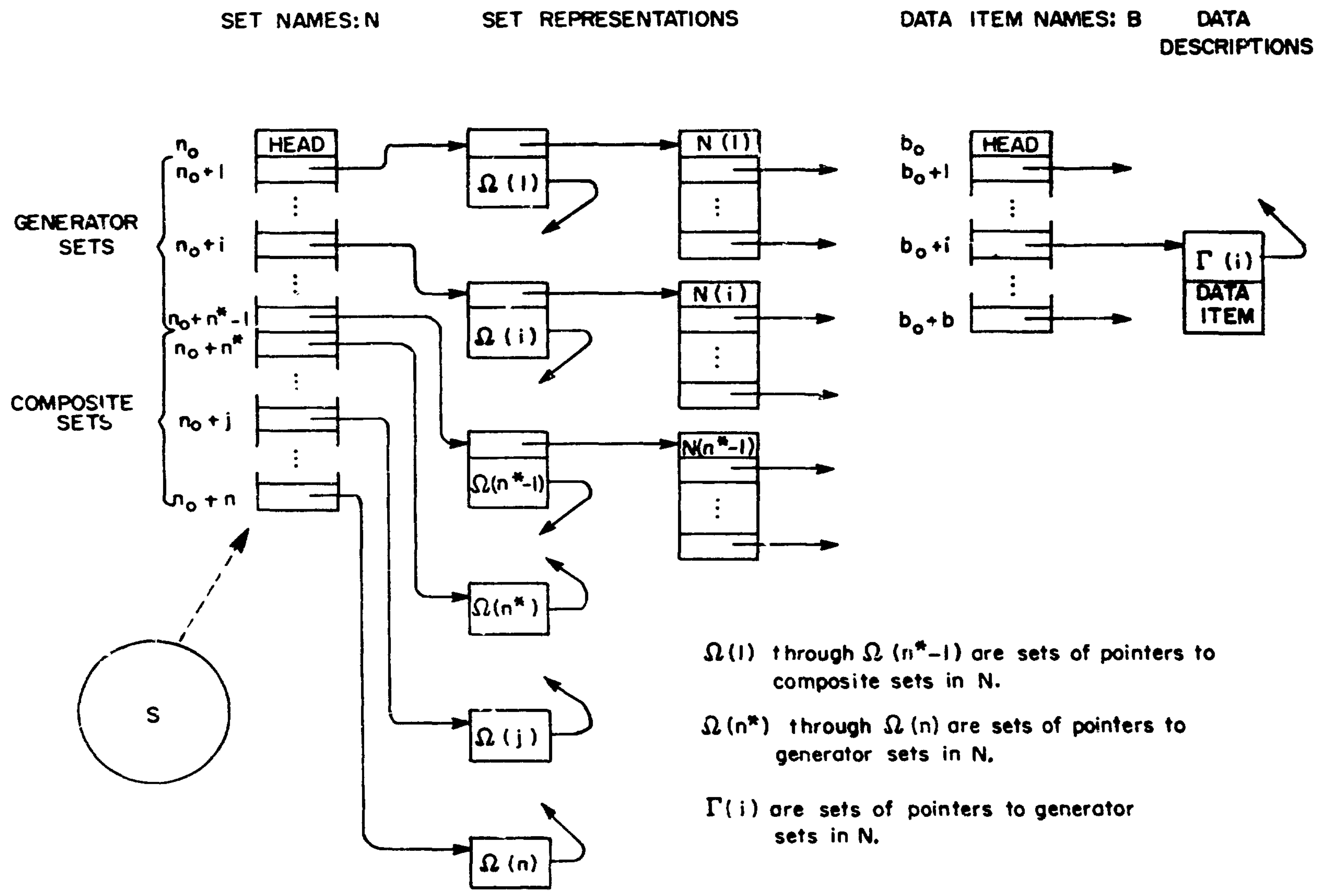
(3) a collection of data item definitions, one for each data item name

(4) a collection N of set names

(5) a collection of set representations, one for each set name

Sets, the collection N of set names, and the collection B of data item names are represented by blocks of contiguous storage locations. The address of a location in the block representing N is a set name and the content of a location in this block is the address of a block representing the corresponding set. Similarly, the address of a location in the block representing B is a data item name and the content of a location in this block is the address of a stored description of the corresponding data item. The blocks which represent individual sets contain the names of the data items which constitute the sets.

In an effort to minimize the storage occupied by the STDS, Childs defines two types of sets: generator sets and composite sets. Only the



$\Omega(1)$ through $\Omega(n^*-1)$ are sets of pointers to composite sets in N.

$\Omega(n^*)$ through $\Omega(n)$ are sets of pointers to generator sets in N.

$\Gamma(i)$ are sets of pointers to generator sets in N.

Figure 3-17. Set-Theoretic Data Structure (Childs)

generator sets have storage representations; the generator sets are disjoint; and the composite sets are unions of generator sets.

With these general characteristics in mind we can examine the STDS in more detail. The block of locations representing the collection B of data item names is assumed to have location $b_o$ as the address of its head. The first location containing a pointer to a data item description has the address $b_o+1$, and the location containing a pointer to the i-th data item description has the address $b_o+i$. If b represents the cardinality of the set B, then the last location containing a pointer to a data item description is $b_c+b$. Since all pointers to data item descriptions are located between $b_o+1$ and $b_o+b$, B may be represented by the set of integers $\{1, 2, \ldots, b\}$. Therefore, any integer i such that $1 \leq i \leq b$ is the data item name for the pointer to the i-th data item description. The pointer to the i-th data item locates a block of storage containing a description of the i-th data item and a list of all generator set names (elements of N) of which the i-th data item is a constituent.

The block of locations representing the collection N of set names is similar to the block representing B, with $n_o$ and n as the address of the head and the cardinality, respectively. The contents of the block representing N are pointers, also. The pointers fall into two classes and are distinguished by an integer n* such that $1 < n^* \leq n$. For all $1 \leq i < n^*$, i is the name of a generator set, and for all $n^* \leq i \leq n$, i is the name of a composite set. A generator set has a set representation, while a composite set does not since it is the union of some generator sets. For $i \geq n^*$ the pointer in $n_0+i$ locates a block of storage containing the names of generator sets, the union of which forms the corresponding composite set. For $i < n^*$ the pointer in $n_0+i$ locates a block of storage containing the names of all composite sets which use generator set i and a pointer to a block of locations containing the names of those data items which form the set.

Let us define our data items and the corresponding data structure for the STDS as follows:

(1) Let N be the collection of set names.

(2) Let M be the collection of sets themselves.

(3) Let M* be the collection of generator set descriptions.

(4) Let B be the collection of data item names.

(5) Let D be the collection of data item descriptions.

(6) Let $r_1$ be a one-to-one relation from N onto M which assigns to every set name in N a unique set in M.

(7) Let $r_2$ be a one-to-one relation from M to M* which assigns to every generator set in M a unique description in M*.

(8) Let $r_3$ be a relation from M to N which assigns to every generator set in M the names in N of those composite sets of which the given generator set is a subset.

(9) Let $r_4$ be a relation from M to N which assigns to every composite set in M the names in N of those generator sets which are subsets of the given composite set.

(10) Let $r_5$ be a relation from M* to B which assigns to every generator set description in M* the names in B of those data items which are elements of the given set.

(11) Let $r_6$ be a one-to-one relation from B <u>onto</u> D which assigns to every data item name in B a unique description in D.

(12) Let $r_7$ be a relation from D to N which assigns to every data item description in D the names in N of those generator sets of which the given data item is an element.

From these definitions we see that

$$P = \{r_j \mid j=1,2,\ldots,7\}$$

$$\Delta = \Pi = N \cup M \cup M^* \cup B \cup D$$

Let us assign the following values to our decision variables:

$\phi_i = \phi_i' = \phi_i'' = 0$ for all $i \epsilon \{1,2,\ldots,10\}$

$\Delta_i = 1$ for all $i \epsilon \{1,2,\ldots,10\}$

$\tau_i = 0$ for all $i \epsilon \{1,2,\ldots,11\}$

$\sigma_1 = 0,\ \sigma_2 = 1$

$\rho_1 = \rho_2 = \rho_3 = 0$

It follows from Constraint 4' that $\beta_i = 1$ for all $i \epsilon \{2,3,\ldots,10\}$.

Assuming that source, relation, and target rings are not used, these decision variable values result in the storage structure of Figure 3-18.

Ignoring for the moment the fact that in the STDS the blocks representing set names are adjacent to one another, as are the blocks representing data item names, it is clear that the STDS is simply the storage structure of Figure 3-18. To elaborate upon this somewhat, let us examine the various blocks of the STDS as shown in Figure 3-17, starting at the left with the blocks representing set names and working to the right through the blocks representing generator and composite sets, generator set descriptions, data item names, and data item descriptions.

Each block representing a set name (i.e., each location in the block of locations representing the set N) is equivalent to the structure of Figure 3-18 for $m=1$. In this case the description block of the SSM is null and the relation of interest is $r_1$.

Each block representing a generator or composite set is also equivalent to the structure of Figure 3-18. The description block of the SSM is again null, but $m \geq 1$. For composite sets the relation of interest is $r_4$. On the other hand, for generator sets there are two relations of interest, namely, $r_2$ and $r_3$. If we assume that the description block indicator for the target of relation $r_2$ (which has only a single target since it is one-to-one) appears first in the stack of description block

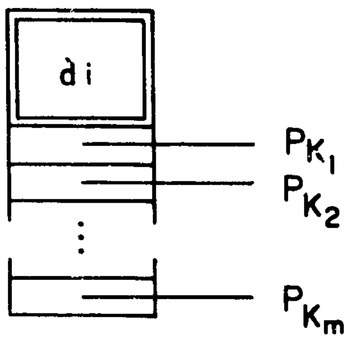


Figure 3-18. Transformed Storage Structure Model - 4

indicators, followed by those for the targets of relation $r_3$, then there is no need of a relation symbol field and we may continue to assume $\rho_1 = \rho_2 = \rho_3 = 0$.

The blocks representing generator set descriptions correspond to the structure of Figure 3-18 for relation $r_5$ and $m \geq 1$.

The situation for the blocks representing data item names is analogous to that for the blocks representing set names and the relation of interest is $r_6$.

Finally, each block representing a data item description is equivalent to the structure of Figure 3-18 for $m \geq 1$ and $r_7$ as the relation of interest.

Let us consider now the adjacency of the blocks representing set names and that of the blocks representing data item names. Since the collection of set names and the collection of data item names act as the (only) entry points into the STDS, we will include (contrary to our earlier assumption) source rings for the set names and the data item names. Because there is no duplication of these names within the STDS, the source rings need not be closed, however. In particular, a source ring need contain only a pointer from the head of that "ring" to the $b_1$-block representing the corresponding source, and the $b_1$-block need not contain a pointer back to the head. Moreover, by stacking all blocks representing set names and by doing the same for all blocks representing data item names and then by assuming (as Childs does) that set names are integers between 1 and n and that data item names are integers between 1 and b, we may implement the source ring "pointers" via a special case of the calculation method of the random data organization described in Chapter I. Given a set name (an integer) or a data item name (an integer), we can determine the (unique) address of the corresponding block simply by adding $n_0$ or $b_0$, respectively, to the name. The result is the STDS described by Childs.

We note that for completeness we may also wish to define two ordering relations $r_8$ and $r_9$ for the collection N of set names and the collection B of data item names, respectively. These relations may be represented within the context of the SSM simply by setting $\sigma_2=0$ and using all other decision variable values as given for the structure of Figure 3-18.

This concludes our discussion of the Storage Structure Model. We have described in this chapter a model for the storage structures capable of representing, via the three basic organizational methods, the intrinsic structure of a given collection of data items as described by the Data Structure Model of Chapter II. The model consists of a given relatively general storage structure (Figure 3-10) together with a number of transformations which may be applied to this storage structure, the combination of which is described by a number of binary-valued decision variables. In addition we have shown the model to be flexible and capable of representing the intrinsic structure of data in varying degrees of explicit detail.

Chapter IV

# ANALYSIS OF THE STORAGE STRUCTURE MODEL

In the previous chapter we developed a model for the storage structures capable of representing the intrinsic structure of some collection of data items as given by the Data Structure Model developed in Chapter II. Our purpose for developing the Storage Structure Model was not simply to illustrate the fact that the intrinsic structure of some given collection of data items can be represented by a multitude of storage structures. Rather it was our purpose to develop a vehicle by means of which we can compare the relative merits of these storage structures.

Our goal in this chapter will be the development of certain measures of performance for the SSM which we can use to make these comparisons.

## 4.1 Measures of Performance

It is generally conceded that the amount of time required to perform certain operations upon a given storage structure and the amount of storage occupied by that structure are the two most important factors to be considered in determining the "goodness" of the structure. One might argue that a factor such as the number of man-hours required for implementation of the software associated with a given storage structure is equally important, but if the structure is to see more than limited use, this factor is of transient interest and, hence, of little importance by comparison.

It is also generally (but not always) true that one may reduce time at the expense of increasing storage and vice versa.

Under many computer operating systems- the batch, monoprogrammed, real-memory systems, for instance - the cost of solving a given problem (i.e., running a given program) is based soley upon the CPU time required to achieve the solution, with the only constraint placed upon the storage being that the program and storage structure must fit into the available storage (usually, quite substantial).

At the other extreme, with time-shared, multi-programmed, virtual-memory systems, the cost of solving a problem is often a function of the amount of storage occupied as well as the CPU time, although for these systems the amount of storage available for a program and its storage structure is virtually unlimited.

In the first case, that storage structure which causes CPU time to be minimized and occupies no more than the storage available is clearly the best. In other words, we are willing to use any amount of storage, up to the maximum amount available, as long as the use of that storage allows CPU time to be reduced.

In the second case, however, no such clear-cut policy exists. Suppose that the cost C of running a program is given by

$$C = k\,[\,T(1+0.01P)]$$

where T represents CPU time, P represents the average number of pages of virtual memory required during the running of the program, and k represents the dollar cost per unit of CPU time. (This is the charging scheme currently in use for MTS, the Michigan Terminal System, at The University of Michigan. [ ]) Increasing by 100 pages the amount of storage used is equivalent to increasing CPU time by T units.

The question we would like to ask is, How much must CPU time be reduced in order to justify an increase of 100 pages in the virtual memory used? Let T represent CPU time for P pages of memory and let T-t represent CPU time for P+100 pages, where t represents the net reduction in CPU time. Then

$$(T-t)[1+.01(P+100)] \leq T(1+.01P)$$

must be true in order to justify the increase in storage. It follows that

$$t \geq \frac{T}{2+.01P}$$

must be true to effect a net gain when storage is increased by 100 pages. Figure 4-1 contains a plot of t/T, the minimum fractional reduction of CPU time required to offset an increase of 100 pages of storage, versus P, the initial storage requirement.

Clearly, a smaller percentage reduction in CPU time is necessary to justify an increase in the storage used if the storage requirement is large to begin with.

We are faced, however, with the problem of not knowing how CPU time and storage will vary as the storage structure is perturbed Moreover, we have considered but a single cost function. We must therefore, develop a procedure for evaluating the time and storage requirements of each storage structure represented by the SSM, while at the same time remaining flexible enough to consider a variety of cost functions.

Basically, the procedure which we will use here is to determine

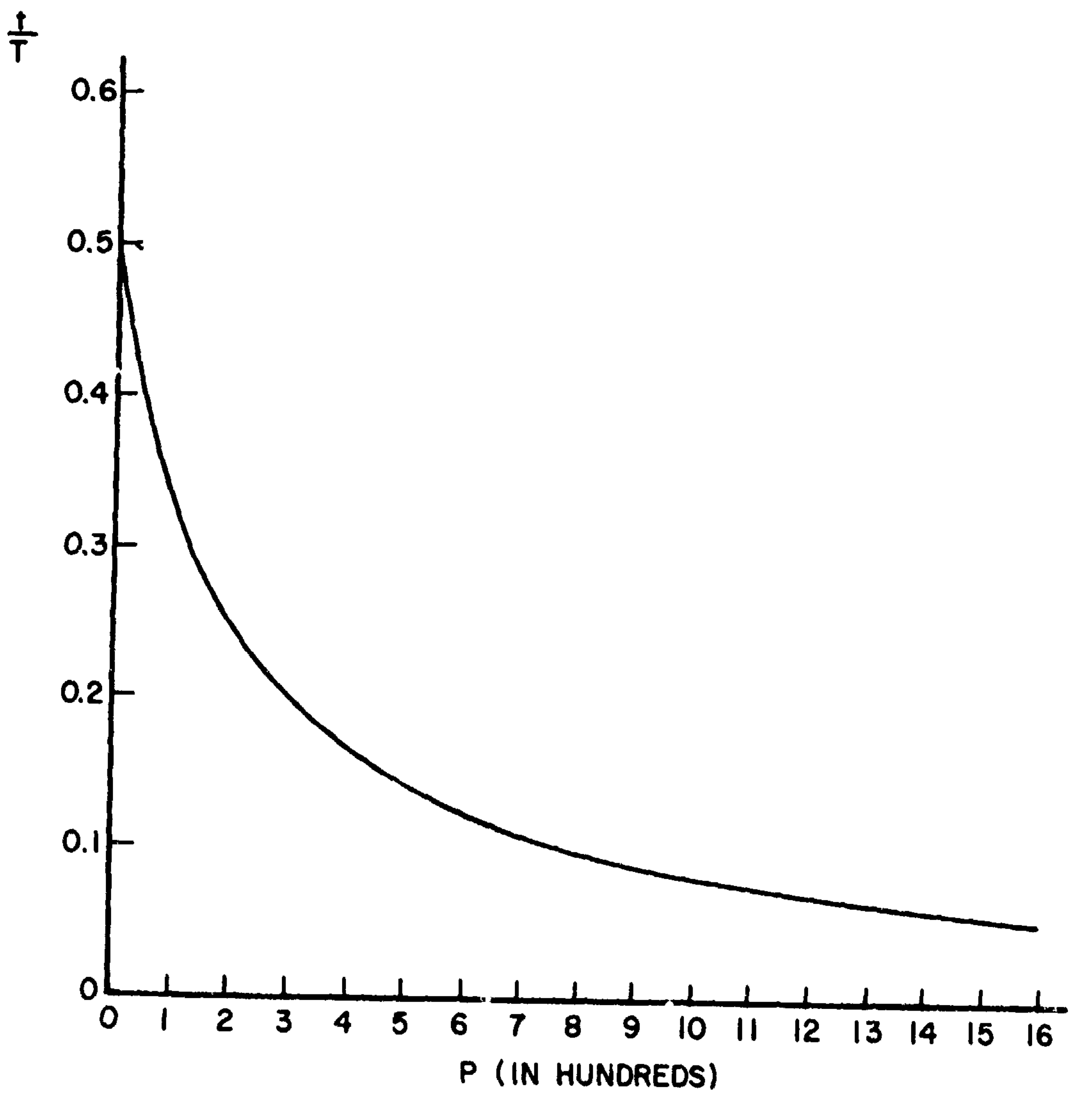
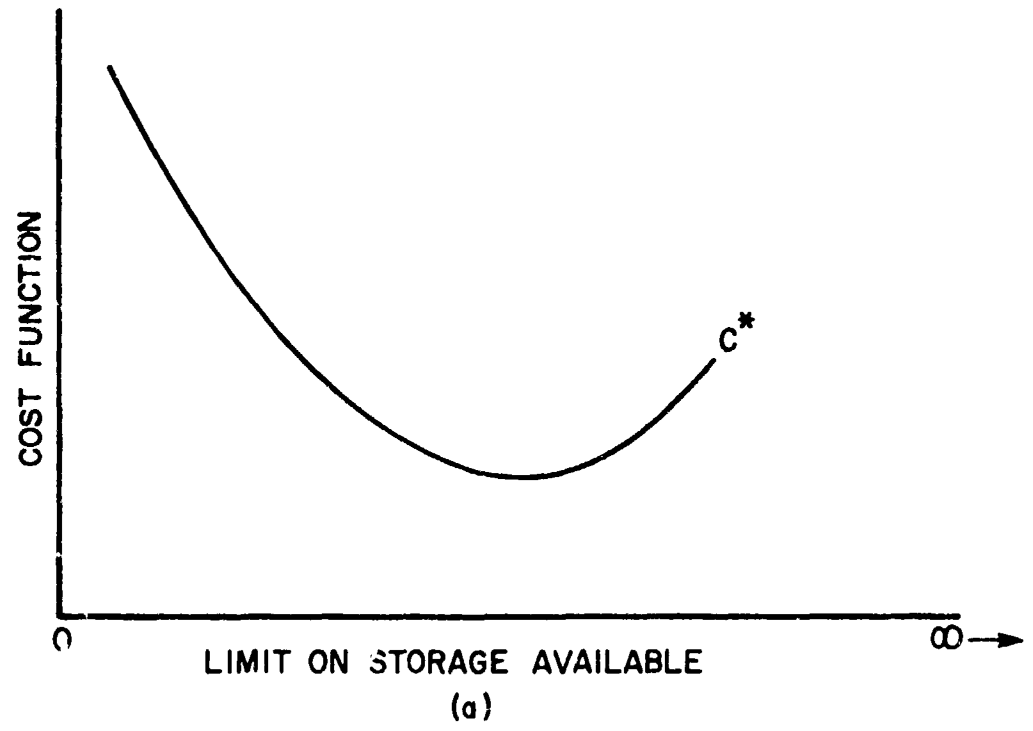


Figure 4-1. Minimum Fractional Reduction of CPU Time Necessary to Offset an Increase of 100 Pages in Program Size for Cost Function C=k[ T(1+0. 01P) ]

the storage structure that minimizes time, subject to a constraint on storage. Clearly, this guarantees an optimal storage structure for the batch, monoprogrammed, real-memory systems.

For systems with more complex cost functions, if we ease the storage constraint to allow for virtually infinite storage, we will be guaranteed a storage structure which is optimal in the sense that it minimizes time, although it may not be optimal in the sense that it minimizes the given cost function. Then using the storage structure determined in this manner as a basis for comparison we may constrain the available storage to be less than that required for our comparison structure and determine the storage structure which minimizes time subject to this constraint. If the storage structure thus determined results in a lower value for the given cost function, we may use it as our comparison structure and repeat the procedure. If the value of the cost function exceeds the best so far, we may ease the storage constraint somewhat and then repeat the procedure.

If the given cost function behaves as shown in Figure 4-2(a), this procedure will allow us to come as close to the optimal solution as we wish, but if the given cost function behaves as shown in Figure 4-2(b), we may in fact determine a solution which is only suboptimal. In each of these figures C* represents the value of the cost function for the storage structure obtained when storage is unconstrained.



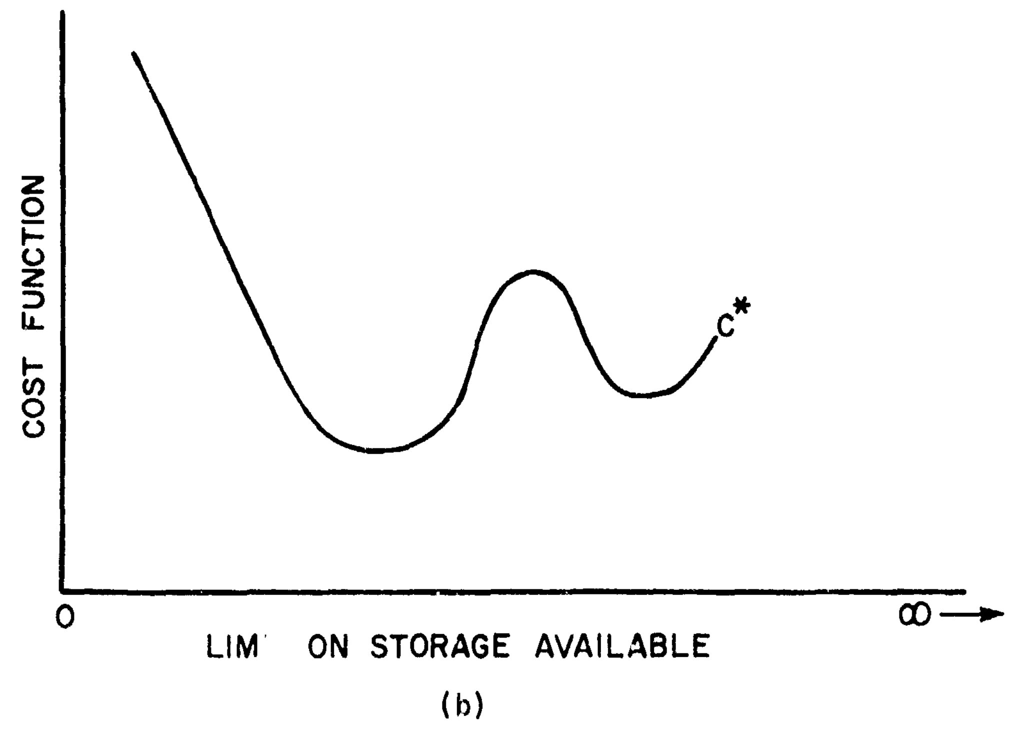


Figure -2. Possible Cost Fu ction Behavior

Since we have no way of determining the behavior of the cost function we can guarantee only a suboptimal solution (provided the process converges). Of course, if we are willing to determine the best solution for every possible storage constraint between 0 and the storage required for our initial solution, we can always guarantee an optimal solution.

Our discussions for the remainder of this chapter will assume simply that we are interested in determining that storage structure which minimizes time subject to a constraint (possibly infinite) on storage.

## 4.2 Time Cost Function

We have determined that in order to compare the relative merits of a collection of storage structures we must characterize each storage structure by two quantities: time and storage.

The storage characteristic of a given storage structure is simply the number of storage units occupied by the structure and, hence, is relatively well defined. The time characteristic of a storage structure is not, however, so well defined.

Intuitively, we feel that the time characteristic should be some measure of the amount of time required to perform certain operations upon the given storage structure. These operations may, of course, vary with the problem being solved.

We will define a number of primitive operations which are representative of the types of operations one might wish to perform upon a collection of data items and from which one can construct other more complex operations. Then in order to define a particular problem which we wish to solve, we will assign weights to each of these primitive operations to reflect the relative frequency with which the operation is used in the solution of the problem.

Let $Q_i$ where $i \epsilon \{1, 2, \cdots, N\}$ represent some primitive operation and let $a_i$ represent the relative frequency of that operation. As a matter of convenience let us place the following constraints upon the various $a_i$:

$$0 \leq a_i \leq 1.0 \quad \text{for } i \epsilon \{1, 2, \cdots, N\}$$

$$\sum_{i=1}^{N} a_i = 1.0$$

Then if we can determine the number of time units $t_i$ required to perform operation $Q_i$, where $i \epsilon \{1, 2, \cdots, N\}$, using a particular storage structure, we will define the <u>time cost</u> T for that storage structure as

$$T = \sum_{i=1}^{N} a_i t_i$$

Assuming no constraint on storage, that storage structure for which T is the smallest will be considered the optimal structure to use in the solution of the given problem (as defined by the various $a_i$).

Thus, we are faced with two problems:

(1) Definition of the primitive operations $Q_i$ for $i \in \{1, 2, \cdots, N\}$.

(2) Determination of the time cost $t_i$ for each primitive operation $Q_i$.

In defining the primitive operations we may separate them into two classes: interrogative operations and manipulative operations. Interrogative operations query a storage structure to determine certain information about the data represented, such as the targets associated with a given source/relation symbol pair. On the other hand, manipulative operations result in alterations to the structure to effect changes in the data represented, such as the addition of a relation instance to the structure.

Although the manipulative operations are no less important, we will confine ourselves to considering only the interrogative operations. The reason for this restriction is simply to narrow the scope of our problem. The techniques which we develop here will be equally applicable to both types of operations. Thus, given sufficient time we could extend our consideration to the manipulative operations as well.

## 4.2.1 Primitive Operations

Since our model for the intrinsic structure of data is based upon the relation instance, it would seem logical to base our primitive operations upon the relation instance, also.

The form of the relation instance is d r p where $d \epsilon \Delta$, $r \epsilon P$, and $p \epsilon \Pi$. We may define our primitive operations by assigning to the three fields of the relation instance form fixed values, variables, and "don't cares" in various combinations.

The operation defined by assigning some particular value to each field of the relation instance form simply determines whether or not the resultant relation instance is true. For instance, if we assign the values $d_i \epsilon \Delta$, $r_j \epsilon P$, and $p_k \epsilon \Pi$ to the three fields of the form (in that order), the resultant triple $d_i\ r_j\ p_k$ may or may not constitute a valid (or true) relation instance. This operation compares the triple $d_i\ r_j\ p_k$ with each known relation instance. If a match is found, the value of the operation is true; otherwise the value is false.

For ease of reference we will characterize this operation by the triple $d_i\ r_j\ p_k$, which we will call its <u>prototype</u>.

Suppose we assign the values $d_i \epsilon \Delta$ and $r_j \epsilon P$ to the first two fields of the form and a variable to the last field, which results in the prototype $d_i\ r_j$ -, where the " - " represents a variable. The operation defined in this manner determines all targets p which satisfy $d_i \; r_j \; p$. We assume that the source $d_i$ is associated with the relation symbol $r_j$. Therefore, this operation will find at least one target satisfying $d_i \; r_j \; p$. We can define two other operations with prototypes $d_i - p_k$ and $- r_j \; p_k$, respectively, which perform similar functions.

Instead of assigning a variable to the third field of the form as above, we might assign a "don't care", which results in the prototype $d_i \; r_j \; *$, where the "*" represents the "don't care". This operation determines whether or not the source $d_i$ is associated with the relation symbol $r_j$ and has a value true or false, accordingly. Again we can define two similar operations with prototypes $d_i * p_k$ and $* \; r_i \; p_k$, respectively.

If we assign a particular value to but a single field of the relation instance form, we have several other possibilities available to us. The prototype $- \; r_j \; -$ defines an operation which determines all source/target pairs (d,p) such that $d \; r_j \; p$ is true. The prototypes $d_i - -$ and $- - p_k$ define similar operations. The prototype $- \; r_j *$ defines an operation which determines all sources d associated with the relation symbol $r_j$. Similar to this operation are those with the following prototypes: $* \; r_j \; -$, $d_i - *$, $d_i * -$, $- * p_k$, and $* - p_k$.

All of these operations are summarized in Table 4-1.

| Operation Code | Prototype | Description |
|---|---|---|
| $Q_1$ | $d_i\ r_j\ p_k$ | Is $d_i\ r_j\ p_k$ true? |
| $Q_2$ | $d_i\ r_j$ * | Is $d_i$ associated with $r_j$? |
| $Q_3$ | $d_i\ r_j$ - | Determine all p such that $d_i\ r_j$ p. |
| $Q_4$ | * $r_j\ p_k$ | Is $p_k$ associated with $r_j$? |
| $Q_5$ | - $r_j\ p_k$ | Determine all d such that d $r_j\ p_k$. |
| $Q_6$ | $d_i$ * $p_k$ | Is $p_k$ associated with $d_i$? |
| $Q_7$ | $d_i$ - $p_k$ | Determine all r such that $d_i$ r $p_k$. |
| $Q_8$ | - $r_j$ * | Determine all d associated with $r_j$. |
| $Q_9$ | * $r_j$ - | Determine all p associated with $r_j$. |
| $Q_{10}$ | - $r_j$ - | Determine all (d, p) such that d $r_j$ p. |
| $Q_{11}$ | $d_i$ - * | Determine all r associated with $d_i$. |
| $Q_{12}$ | $d_i$ * - | Determine all p associated with $d_i$. |
| $Q_{13}$ | $d_i$ - - | Determine all (r, p) such that $d_i$ r p. |
| $Q_{14}$ | - * $p_k$ | Determine all d associated with $p_k$. |
| $Q_{15}$ | * - $p_k$ | Determine all r associated with $p_k$. |
| $Q_{16}$ | - - $p_k$ | Determine all (d, r) such that d r $p_k$. |

* indicates "don't care"
- indicates variable field

Table 4-1. Primitive Operations

We could have defined an operation with the prototype ---, that is, one with variables in all positions of the relation instance form, but such an operation would provide all the relation instances of the structure and is, therefore, not very enlightening. Similarly, we could have defined operations with the prototypes $d_i$**, *$r_j$*, and **$p_k$, respectively, but since we assume that $d_i$, $r_j$, and $p_k$ must appear somewhere in the structure, these operations provide no information at all.

Clearly, we still have not exhausted all the possibilities for operations which may be defined using the form d r p. We have certainly considered all possible operations for which the fixed quantities are constrainted to be single-valued, but we could define similar operations for which the fixed quanities are allowed to be mutli-valued. For instance, we could define an operation which determines all targets shared by some <u>set</u> of source/relation symbol pairs. We choose not to do this, however, for the same reason we are not considering manipulative operations. Nonetheless, the sixteen primitive operations which we have defined should be representative of most (interrogative) operations we might want to consider.

We are now faced with the problem of determining the time cost $t_i$ for each primitive operation $Q_i$ where $i \in \{1, 2, \cdots, 16\}$.

But in order to determine $t_i$ we need some algorithm or procedure which describes the implementation of operation $Q_i$. Clearly, there does not exist any such unique procedure.

Consider for example operation $Q_1$: $d_i$ $r_j$ $p_k$. For this operation we may define four basic procedures:

(1) Find $d_i$ in the storage structure. Examine all relation symbols associated with $d_i$ in search of $r_j$. If $r_j$ is found, examine all targets in the corresponding target set in search of $p_k$.

(2) Find $r_j$ (the relation, not just a relation symbol) in the storage structure. Examine all sources associated with $r_j$ in search of $d_i$. If $d_i$ is found, examine all targets in the corresponding target set in search of $p_k$.

(3) Find $r_j$ in the storage structure. Examine all targets associated with $r_j$ in search of $p_k$. If $p_k$ is found, examine all sources associated with the appropriate relation symbol/target set pair in search of $d_i$.

(4) Find $p_k$ in the storage structure. Examine all relation symbols associated with $p_k$ in search of $r_j$. If $r_j$ is found, examine all sources associated with the appropriate relation symbol-target set pair in search of $d_i$.

Even each of these basic procedures is subject to some variation. For instance, the process of searching for $r_j$ given $d_i$ can be implemented in a number of different ways (although the differences may be slight). However, for our purposes we will assume that each of the basic procedures has some unique implementation. We will choose one which we feel is representative of all the implementations of the given basic procedure and let that implementation be the "unique" one.

Each of our primitive operations will then be characterized by a number of "unique" basic procedures which we will call <u>methods</u>. Operation $Q_1(d_i\ r_j\ p_k)$ is characterized by four methods, for instance. Descriptions of the methods for each of the primitive operations are contained in Appendix A.

Let us introduce a simple notational scheme for describing the methods of an operation. Let us denote the steps in a method by a sequence of symbols, one for each step, from left to right. Let the steps "Search for the source $d_i$", "Search for the relation (or relation symbol) $r_j$", and "Search for the target $p_k$" be denoted by the symbols $\hat{d}$, $\hat{r}$, and $\hat{p}$, respectively. Let the steps "Determine all sources", "Determine all relations (or relation symbols)", and "Determine all targets" be denoted by the symbols d, r, and p, respectively. The sequence $\hat{d}\ r\ \hat{p}$ would then be interpreted as: (1) Search for the source $d_i$, (2) Determine all relation symbols (associated with the source $d_i$), and (3) Search for the target $p_k$ (among those targets associated with the source $d_i$ and the relation symbol currently under scrutiny). Using this notation we have summarized the methods for each of the primitive operations in Table 4-2.

In general, each of the methods of a given primitive operation will have a different time cost. Let the time cost for method j of operation $Q_i$ be represented by $t_{ij}$. We will then define the time cost $t_i$ of operation $Q_i$ by

$$t_i = \min_j t_{ij} \qquad \begin{matrix} i \epsilon \{1, 2, \ldots, 16\} \\ j \epsilon \{1, 2, \ldots, s_i\} \end{matrix}$$

where $s_i$ represents the number of methods for operation $Q_i$.

Thus, to determine the time cost $t_i$ for operation $Q_i$, we must determine the time cost $t_{ij}$ for each method of the operation and then choose that method which has the smallest time cost to represent the operation

## 4.2.2 Elementary Time Costs

In performing a given primitive operation (via any of its methods) upon a storage structure we will find it necessary to trace our way from one point in the structure to another. In general, this will give involve following pointers in rings and sequencing through stacks. In order to determine the time cost of the operation we may,

| Operation | Method 1 | Method 2 | Method 3 | Method 4 |
|---|---|---|---|---|
| $Q_1$: $d_i r_j p_k$ | $\hat{d}\,\hat{r}\,\hat{p}$ | $\hat{r}\,\hat{d}\,\hat{p}$ | $\hat{r}\,\hat{p}\,\hat{d}$ | $\hat{p}\,\hat{r}\,\hat{d}$ |
| $Q_2$: $d_i r_j *$ | $\hat{d}\,\hat{r}$ | $\hat{r}\,\hat{d}$ | | |
| $Q_3$: $d_i r_j -$ | $\hat{d}\,\hat{r}\,p$ | $\hat{r}\,\hat{d}\,p$ | $p\,\hat{r}\,\hat{d}$ | |
| $Q_4$: $* r_j p_k$ | $\hat{r}\,\hat{p}$ | $\hat{p}\,\hat{r}$ | | |
| $Q_5$: $- r_j p_k$ | $\hat{r}\,\hat{p}\,d$ | $\hat{p}\,\hat{r}\,d$ | $d\,\hat{r}\,\hat{p}$ | |
| $Q_6$: $d_i * p_k$ | $\hat{d}\,\hat{p}$ | $\hat{p}\,\hat{d}$ | | |
| $Q_7$: $d_i - p_k$ | $\hat{d}\,r\,\hat{p}$ | $\hat{p}\,r\,\hat{d}$ | $r\,\hat{d}\,\hat{p}$ | $r\,\hat{p}\,\hat{d}$ |
| $Q_8$: $- r_j *$ | $\hat{r}\,d$ | $d\,\hat{r}$ | | |
| $Q_9$: $* r_j -$ | $\hat{r}\,p$ | $p\,\hat{r}$ | | |
| $Q_{10}$: $- r_j -$ | $\hat{r}\,d\,p$ | $\hat{r}\,p\,d$ | $d\,\hat{r}\,p$ | $p\,\hat{r}\,d$ |
| $Q_{11}$: $d_i - *$ | $\hat{d}\,r$ | $r\,\hat{d}$ | | |
| $Q_{12}$: $d_i * -$ | $\hat{d}\,p$ | $p\,\hat{d}$ | | |
| $Q_{13}$: $d_i - -$ | $\hat{d}\,r\,p$ | $r\,\hat{d}\,p$ | | |
| $Q_{14}$: $- * p_k$ | $\hat{p}\,d$ | $d\,\hat{p}$ | | |
| $Q_{15}$: $* - p_k$ | $\hat{p}\,r$ | $r\,\hat{p}$ | | |
| $Q_{16}$: $- - p_k$ | $\hat{p}\,r\,d$ | $r\,\hat{p}\,d$ | | |

Table 4-2. Summary of Methods of Implementation for Primitive Operations

therefore, wish to know the amount of time required to access one block in the structure from another. As a result, we will introduce a number of elementary time costs, each of which represents the amount of time required to access one particular type of block in the storage structure from another.

Before beginning consideration of these elementary time costs, let us clarify somewhat the concept of tracing through a storage structure. We assume that there exists a position indicator, or simply an indicator, which contains the address of that field (in some block) currently of interest or under consideration. The indicator is clearly just a pointer to the field of interest. As our interest shifts from point to point within the structure, the value of the indicator changes to reflect this. More correctly, the value of the indicator changes to reflect the effects of operations applied to the structure to elicit information from it, and our interest shifts accordingly. Tracing through a storage structure simply amounts to stepping the indicator along certain paths of access within the structure as required by operations applied to the structure.

Let I represent the current value of the indicator. I is then equal to the address of the field of current interest. Let (I) represent the contents, or value, of the field whose address is I.

We will now define a number of very basic quantities to be used in the formulation of the elementary time cost expressions. Let $f_i$

represent the number of time units required to follow a forward pointer from one block to another in an $a_i$-ring of the SSM, where $i \epsilon \{1, 2, \cdots, 10\}$. $f_i$ is then the number of time units required to replace I by (I), where I is the address of a forward pointer field in some block of an $a_i$-ring. $f_i$ is of interest, of course, only if $\phi_i = 1$.

Let $s_i$ represent the number of time units required to step or sequence from one block to another through a stack of blocks in an $a_i$-ring of the SSM, where $i \epsilon \{1, 2, \cdots, 10\}$. $s_i$ is then the number of time units required to add (or subtract) some displacement D to I, where I is the address of an arbitrary field in some block of an $a_i$-ring. Clearly, $s_i$ is of interest only if $\phi_i = 0$ and $\Delta_i = 0$.

Let $h_i$ represent the number of time units required to follow a head pointer from an element block to the head of an $a_i$-ring of the SSM, where $i \epsilon \{1, 2, \cdots, 10\}$. $h_i$ is then the number of time units required to replace I by (I), where I is the address of a head pointer field in some element block of an $a_i$-ring. Of course, $h_i$ is of interest only if $\phi'_i = 1$.

Finally, let $s_0$ represent the number of time units required to step from one pointer field of a given block to another pointer field of that block. For example, in tracing through a structure we might enter a particular block via one ring and desire to continue our tracing with the second ring which passes through the block. In order to do this, we must add (or subtract) some displacement D to I, where I is the address of the pointer field in the first ring and $I \pm D$ is the address of the pointer field in the second ring. $s_0$ reflects the time required to carry out this addition (or subtraction). Note that the second pointer field may contain a head pointer instead of a ring (forward) pointer, but the process to be carried out is essentially the same.

To be completely general we should also define some quantity to represent the number of time units required to follow a reverse pointer in each of the rings of the SSM. However, reverse pointers are generally useful only for the manipulative operations which we might define and since we are not considering manipulative operations (and since reverse pointers will not be useful for our interrogative operations), we shall assume that the structures we consider contain no reverse pointers (i.e., $\phi''_i = 0$ for all $i \in \{1, 2, \cdots, 10\}$). Hence, we have no need for the quantity mentioned.

We might question the usefulness of defining a separate quantity $f_i$ (or $s_i$ or $h_i$) for each ring of the SSM. There are two reasons for doing so. The first is for the sake of generality, and the second is for convenience when it comes time to develop expressions for the elementary time costs (we can tell at a glance which $a_i$-rings of the SSM are under consideration).

Realistically, it would be quite unusual if $s_0, s_1, s_2, \cdots, s_{10}$ were not all equal. The same may in general be said of $f_i$ and $h_i$.

In fact, $f_i$ and $h_i$ are probably equal to one another for all values of i, also. It is possible, however, for pointers to be stored in different size fields or for them to be located in different relative portions of a storage unit or for some to require shifting before use and so forth. All of these factors can contribute to differences in the times arising from their use. Furthermore, instead of actual pointers we may want random pointers in some rings. As we indicated earlier, the uses of these two types of pointers will in general be characterized by different amounts of time.

We will make two assumptions regarding the various quantities $f_i$, $s_i$, and $h_i$. First we will assume that the time required to follow a pointer is always at least as great as the time to step through a stack. Thus, $s_i \leq f_i$ and $s_i \leq h_i$ for $i \epsilon \{1, 2, \cdots, 10\}$. Since following a pointer will always involve a storage reference whereas stepping through a stack need not, this is a reasonable assumption to make. Second, we will assume that the time required to follow a head pointer will never be less than the time to follow a forward pointer. Hence, $f_i \leq h_i$ for $i \epsilon \{1, 2, \cdots, 10\}$. This implies that if either the forward pointer or the head pointer must be subject to additional processing (because of different size fields, etc.) the head pointer will be chosen for this additional processing. Since forward pointers will in general be used more frequently than head pointers (when both appear in the same ring), this is a reasonable assumption to make. Combining these two assumptions, we can write

$$s_i \leq f_i \leq h_i \quad \text{for } i \in \{1, 2, \cdots, 10\}.$$

(We note that this assumption is not crucial to the development of the time cost function, but reflects simply a condition which generally exists.)

Let us return now to consideration of the elementary time costs in which we have expressed some interest but which we have not yet defined. These elementary time costs will be divided into three classes as follows: (1) those which reflect the number of time units required to move (i.e., trace our way) from one element block of a ring of the SSM to another element block of that ring, (2) those which reflect the number of time units required to move from an (arbitrary) element block of a ring to the head of that ring, and (3) those which reflect the number of time units required to move from the head of a ring to the first element block of that ring or vice versa.

Let $e_i$ represent the number of time units required to move from one element block of an $a_i$-ring of the SSM to another element block of that ring, where $i \in \{1, 2, \cdots, 10\}$.

Consider for example $e_2$, the number of time units required to move from one $b_3$-block in an $a_2$-ring of the SSM to another $b_3$-block in that ring.

If $\phi_2 = 1$, the $a_2$-rings are indeed explicit rings and we need only follow a forward pointer to move from one $b_3$-block to another. Hence, in this case $e_2 = f_2$.

On the other hand if $\phi_2 = 0$, there exist a number of possibilities. If $\Delta_2 = 1$, each $a_2$-ring (which is not really a ring, of course) may be viewed as containing a single $b_3$-block. We will assume, therefore, that $e_2 = 0$.

If $\Delta_2 = 0$, we are again faced with a number of alternatives. Suppose $\phi_3 = 1$. Then the $b_3$-blocks of a given $a_2$-ring will be stacked upon the corresponding $b_2$-block, and $e_2 = s_2$.

A more complex situation could arise if instead $\phi_3 = 0$, $\phi_4 = 0$, $\phi_5 = 1$, $\Delta_3 = 1$, and $\Delta_4 = 0$, which results in the situation depicted in Figure 4-3 (where type fields have been included in each block for clarity). In this case, the $b_5$-blocks of each $a_4$-ring are stacked upon the corresponding $b_4$-block, which is duplicated upon each $b_3$-block of the associated $a_3$-ring. The $b_3$-blocks of each $a_2$-ring are in turn stacked upon the corresponding $b_2$-block.

Since it is possible for the $b_3$-blocks or the $b_5$-blocks or both to contain head pointers to $b_2$-blocks and $b_4$-blocks, respectively, and since these head pointers may be used to advantage if we wish to move up the structure for Figure 4-3, it now becomes important to know whether we are moving from some $b_3$-block to the $b_3$-block

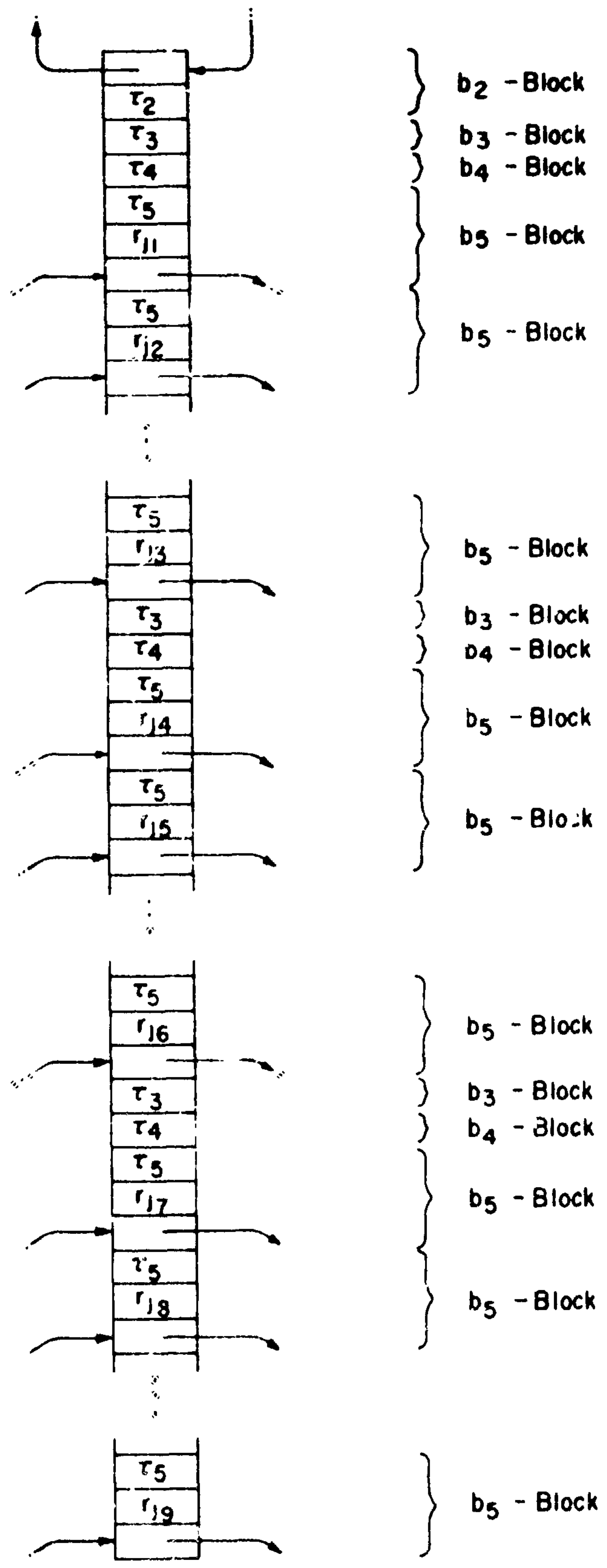


Figure 4-3. Example of Stacking-Duplication-Stacking

above it or to the $b_3$-block below it. This will be determined by the general direction in which we are moving through the storage structure as a whole. For instance, if we are tracing through the structure from a $b_1$-block representing some source in search of some $b_6$-block representing a relation symbol, we are moving downward from a given $b_3$-block to the one below it, and vice versa.

Let $\delta_i$ be a binary-valued variable, the value of which indicates whether movement is toward ($\delta_i$=1) or away from ($\delta_i$=0) the head of an $a_i$-ring when the element blocks of that ring are stacked upon the head, where $i \epsilon \{1, 2, \cdots, 10\}$.

Returning to consideration of $e_2$, if $\delta_2 = 1$, we can take advantage of the head pointers (if present) in the stack of $b_5$-blocks between consecutive $b_3$-blocks. Thus, if $\delta_2 = 1$ and $\phi'_4 = 1$, we can step from a $b_3$-block to the first $b_5$-block on the bottom of the stack above it and then follow the head pointer in the $b_5$-block to the $b_4$-block which acts as the head of the stack. Since there is a $b_3$-block associated with every $b_4$-block in the structure, we may treat the two as indistinguishable (i.e., as a single block). Following the head pointer then brings us to the $b_3$-block desired. It follows that $e_2$ is given by $s_2 + h_4$.

If on the other hand $\delta_2 = 0$ or $\phi'_4 = 0$, we must step from the $b_3$-block to the $b_5$-block adjacent to it and then step through the stack of $b_5$-blocks of which there are $k_4$. In this case $e_2 = s_2 + k_4 s_4$.

Thus, if $\phi_2 = 0$, $\phi_3 = 0$, $\phi_4 = 0$, $\phi_5 = 1$, $\Delta_2 = 0$, $\Delta_3 = 1$, and $\Delta_4 = 0$, then $e_2$ will be given by the following expression:

$$e_2 = s_2 + \phi'_4 \delta_2 h_4 + (\overline{\phi}'_4 + \overline{\delta}_2) k_4 s_4$$

where the "+" between the two decision variables $\overline{\phi}'_4$ and $\overline{\delta}_2$ is treated essentially as disjunction. That is, if either $\overline{\phi}'_4$ or $\overline{\delta}_2$ or both are 1, then the value of $\overline{\phi}'_4 + \overline{\delta}_2$ is 1; otherwise its value is 0. We may also treat the product $\phi'_4 \delta_2$ as conjunction, although both the arithmetic product and the logical product yield the same result.

To be consistent, we will treat products and sums of deciscion variables (which are represented by Greek letters) as conjunction and disjunction, respectively. The result of these operations will be either a 1 or a 0 which will then be treated simply as an integer in any arithmetic operation (specifically, an arithmetic product).

For example , in the above expression for $e_2$ the term $\phi'_4 \delta_2 h_4$ is effectively the arithmetic product of $h_4$ and a 1 or a 0, depending upon the result of the logical product $\phi'_4 \delta_2$.

Returning once again to our consideration of $e_2$, suppose $\phi_2=\phi_3=\phi_4=\phi_5=\phi_6=0$, $\phi_7=1$, $\Delta_2=\Delta_4=\Delta_6=0$, and $\Delta_3=\Delta_5=1$. The resultant storage structure will be similar to that of Figure 4-3 but will have stacks of $b_7$-blocks inserted between the various $b_5$-blocks. By applying reasoning like that used above, we generate the following expression for $e_2$:

$$\begin{aligned} e_2 = s_2 &+ \phi'_4 \phi'_6 \delta_2 (h_4 + h_6) \\ &+ \phi'_4 \overline{\phi}'_6 \delta_2 (h_4 + k_6 s_6) \\ &+ \overline{\phi}'_4 \phi'_6 \delta_2 k_4 (s_4 + h_6) \\ &+ (\overline{\phi}'_4 \overline{\phi}'_6 + \overline{\delta}_2) k_4 (s_4 + k_6 s_6) \end{aligned}$$

Implicit in this expression is the assumption that if both the $b_5$-blocks and the $b_7$-blocks contain head pointers. then a head pointer in a $b_7$-block points to the head pointer in the $b_5$-block associated with the $b_6$-block which is the (actual) head of the $a_6$-ring of which the $b_7$-block is a member. We can make this assumption since there is a one-to-one correspondence between the $b_5$-blocks and the $b_6$-blocks and we treat the two types of blocks essentially as one. If we did not make this assumption, we would have to step the indicator from the location pointed to by the $b_7$-block head pointer to that location containing the $b_5$-block head pointer, resulting in an additional time cost $s_o$ to be added to $h_6$.

We can now continue in this manner until we have considered all combinations of the deciscions variables which affect $e_2$.

Since expressions of the sort encountered above occur very frequently in deriving expressions for the various $e_i$, we will define a number of general forms as follows:

$$S_1(j, i_1) = s_j + \phi'_{i_1} \delta_j h_{i_1} + (\overline{\phi}'_{i_1} + \overline{\delta}_j) k_{i_1} s_{i_1}$$

$$S_2(j,i_1,i_2) = s_j + \phi'_{i_1}\phi'_{i_2}\delta_j(h_{i_1}+h_{i_2})$$
$$+\phi'_{i_1}\bar{\phi}'_{i_2}\delta_j(h_{i_1}+k_{i_2}s_{i_2})$$
$$+\bar{\phi}'_{i_1}\phi'_{i_2}\delta_j k_{i_1}(s_{i_1}+h_{i_2})$$
$$+(\bar{\phi}'_{i_1}\bar{\phi}'_{i_2}+\delta_j)k_{i_1}(s_{i_1}+k_{i_2}s_{i_2})$$

$$S_3(j,i_1,i_2,i_3) = s_j+\phi'_{i_1}\phi'_{i_2}\phi'_{i_3}\delta_j(h_{i_1}+h_{i_2}+h_{i_3})$$
$$+\phi'_{i_1}\phi'_{i_2}\bar{\phi}_{i_3}\delta_j(h_{i_1}+h_{i_2}+k_{i_3}s_{i_3})$$
$$+\phi'_{i_1}\bar{\phi}'_{i_2}\phi'_{i_3}\delta_j[h_{i_1}+k_{i_2}(s_{i_2}+h_{i_3})]$$
$$+\phi'_{i_1}\bar{\phi}'_{i_2}\bar{\phi}'_{i_3}\delta_j[h_{i_i}+k_{i_2}(s_{i_2}+k_{i_3}s_{i_3})]$$
$$+\bar{\phi}'_{i_1}\phi'_{i_2}\phi'_{i_3}\delta_j k_{i_1}(s_{i_1}+h_{i_2}+h_{i_3})$$
$$+\bar{\phi}'_{i_1}\phi'_{i_2}\bar{\phi}'_{i_3}\delta_j k_{i_1}(s_{i_1}+h_{i_2}+k_{i_3}s_{i_3})$$
$$+\bar{\phi}'_{i_1}\bar{\phi}'_{i_2}\phi'_{i_3}\delta_j k_{i_1}[s_{i_1}+k_{i_2}(s_{i_2}+h_{i_3})]$$
$$+(\bar{\phi}'_{i_1}\bar{\phi}'_{i_2}\bar{\phi}'_{i_3}+\bar{\delta}_j)k_{i_1}[s_{i_1}+k_{i_2}(s_{i_2}+k_{i_3}s_{i_3})]$$

$$S_4(j,i_1,i_2,i_3,i_4) = s_j + \phi'_{i_1}\phi'_{i_2}\phi'_{i_3}\phi'_{i_4}\delta_j(h_{i_1}+h_{i_2}+h_{i_3}+h_{i_4})$$

$$+\phi'_{i_1}\phi'_{i_2}\phi'_{i_3}\bar{\phi}'_{i_4}\delta_j(h_{i_1}+h_{i_2}+h_{i_3}+k_{i_4}s_{i_4})$$

$$+\phi'_{i_1}\phi'_{i_2}\bar{\phi}'_{i_3}\phi'_{i_4}\delta_j[h_{i_1}+h_{i_2}+k_{i_3}(s_{i_3}+h_{i_4})]$$

$$+\phi'_{i_1}\phi'_{i_2}\bar{\phi}'_{i_3}\bar{\phi}'_{i_4}\delta_j[h_{i_1}+h_{i_2}+k_{i_3}(s_{i_3}+k_{i_4}s_{i_4})]$$

$$+\phi'_{i_1}\bar{\phi}'_{i_2}\phi'_{i_3}\phi'_{i_4}\delta_j[h_{i_1}+k_{i_2}(s_{i_2}+h_{i_3}+h_{i_4})]$$

$$+\phi'_{i_1}\bar{\phi}'_{i_2}\phi'_{i_3}\bar{\phi}'_{i_4}\delta_j[h_{i_1}+k_{i_2}(s_{i_2}+h_{i_3}+k_{i_4}s_{i_4})]$$

$$+\phi'_{i_1}\bar{\phi}'_{i_2}\bar{\phi}'_{i_3}\phi'_{i_4}\delta_j\{h_{i_1}+k_{i_2}[s_{i_2}+k_{i_3}(s_{i_3}+h_{i_4})]\}$$

$$+\phi'_{i_1}\bar{\phi}'_{i_2}\bar{\phi}'_{i_3}\bar{\phi}'_{i_4}\delta_j\{h_{i_1}+k_{i_2}[s_{i_2}+k_{i_3}(s_{i_3}+k_{i_4}s_{i_4})]\}$$

$$+\bar{\phi}'_{i_1}\phi'_{i_2}\phi'_{i_3}\phi'_{i_4}\delta_j k_{i_1}(s_{i_1}+h_{i_2}+h_{i_3}+h_{i_4})$$

$$+\bar{\phi}'_{i_1}\phi'_{i_2}\phi'_{i_3}\bar{\phi}'_{i_4}\delta_j k_{i_1}(s_{i_1}+h_{i_2}+h_{i_3}+k_{i_4}s_{i_4})$$

$$+\bar{\phi}'_{i_1}\phi'_{i_2}\bar{\phi}'_{i_3}\phi'_{i_4}\delta_j k_{i_1}[s_{i_1}+h_{i_2}+k_{i_3}(s_{i_3}+h_{i_4})]$$

$$+\bar{\phi}'_{i_1}\phi'_{i_2}\bar{\phi}'_{i_3}\bar{\phi}'_{i_4}\delta_j k_{i_1}[s_{i_1}+h_{i_2}+k_{i_3}(s_{i_3}+k_{i_4}s_{i_4})]$$

$$+\bar{\phi}'_{i_1}\bar{\phi}'_{i_2}\phi'_{i_3}\phi'_{i_4}\delta_j k_{i_1}[s_{i_1}+k_{i_2}(s_{i_2}+h_{i_3}+h_{i_4})]$$

$$+\bar{\phi}'_{i_1}\bar{\phi}'_{i_2}\phi'_{i_3}\bar{\phi}'_{i_4}\delta_j k_{i_1}[s_{i_1}+k_{i_2}(s_{i_2}+h_{i_3}+k_{i_4}s_{i_4})]$$

$$+\bar{\phi}'_{i_1}\bar{\phi}'_{i_2}\bar{\phi}'_{i_3}\phi'_{i_4}\delta_j k_{i_1}\{s_{i_1}+k_{i_2}[s_{i_2}+k_{i_3}(s_{i_3}+h_{i_4})]\}$$

$$+(\bar{\phi}'_{i_1}\bar{\phi}'_{i_2}\bar{\phi}'_{i_3}\bar{\phi}'_{i_4}+\bar{\delta}_j)k_{i_1}\{s_{i_1}+k_{i_2}[s_{i_2}+k_{i_2}(s_{i_3}+k_{i_4}s_{i_4})]\}$$

We now have a convenient shorthand notation. Instead of having to write

$$e_2 = s_2 + \phi'_4 \delta_2 h_4 + (\bar{\phi}'_4 + \bar{\delta}_2)\, k_4 s_4$$

we can write simply

$$e_2 = S_1(2, 4)$$

Similarly, instead of

$$\begin{aligned} e_2 = s_2 &+ \phi'_4\, \phi'_6\, \delta_2\, (h_4 + h_6) \\ &+ \phi'_4\, \bar{\phi}'_6\, \delta_2\, (h_4 + k_6 h_6) \\ &+ \bar{\phi}'_4 \phi'_6\, \delta_2\ k_4 (s_4 + h_6) \\ &+ (\bar{\phi}'_4\, \bar{\phi}'_6 + \delta_2)\ k_4 (s_4 + k_6 s_6) \end{aligned}$$

we can write

$$e_2 = S_2(2, 4, 6)$$

This notation has been employed to compile a complete tabulation of $e_i$ for all $i \epsilon \{1, 2, \cdots, 10\}$, which appears in Appendix B.

We have implicitly assumed in the derivation of these expressions for $e_i$ that either $\phi_1 = 1$ or $\phi_{10} = 1$ (or both) when both $\sigma_1 = 0$ and $\sigma_2 = 0$. This guarantees, of course, that Constraints 6 and 7 will be satisfied.

There are two reasons for making this assumption. First, this assumption allows us to use the same expressions for the various $e_i$ when $\sigma_1 = 0$ and $\sigma_2 = 0$ as when either $\sigma_1 = 1$ or $\sigma_2 = 1$ or both. (The $b_1$-blocks and the $b_{11}$-blocks then act as "stops" in the structure.)

Second, we need not concern ourselves with developing expressions for the $e_i$ for the case in which $\Delta$ and $\Pi$ are disjoint or the special cases in which $\Delta$ and $\Pi$ are not disjoint. Suppose for instance that $\phi_i = 0$ for all $i \epsilon \{1, 2, \cdots, 10\}$, $\Delta$ and $\Pi$ are not disjoint, and $\sigma_1 = \sigma_2 = 0$. If the intrinsic structure of our data is such that $d_1 r_1 d_2$, $d_2 r_2 d_3$, $d_3 r_3 d_4, \cdots, d_i r_i d_{i+1}, d_{i+1}\ r_{i+1}\ d_{i+2}, \cdots, d_{n-1} r_{n-1} d_n$ (and $d_{i_1} r_{j_1} d_{i_2}$ and $d_{i_2} r_{j_2} d_{i_1}$ are not both true), then clearly our expressions for the various $e_i$ must be a function of n, which item of information is not at our disposal.

Basically then, our reason for making this assumption is to avoid deriving expressions for the $e_i$ to cover each of several possible special cases.

In order to specify the value of $\delta_i$ to be assumed for a particular use of $e_i$, we will follow the convention that a prime is affixed to $e_i$ whenever $\delta_i = 0$. Thus,

$$e_i \text{ implies } \delta_i = 1 \text{ for } i \epsilon \{1, 2, \cdots, 10\}$$
$$e_i' \text{ implies } \delta_i = 0 \text{ for } i \epsilon \{1, 2, \cdots, 10\}.$$

We will continue to use $e_i$ in the generic sense, also, but context should make the usage clear. As a rule of thumb, the value of $\delta_i$ is of significance only when $e_i$ appears in a time expression. Unless otherwise specified, all other uses of $e_i$ are in the generic sense.

The quantity $e_i$ reflects the number of time units required to move from one element block in an $a_i$-ring to another element block in that ring, but in general it does not reflect the number of time units required to move from the element block nearest the head to the head or from the head to that element block.

Let $e_i^o$ represent the number of time units required to move from the element block nearest the head in an $a_i$-ring of the SSM to the head of that ring or from the head to that element block, where $i \epsilon \{1, 2, \cdots, 10\}$. Note that if $\phi_i = 1$ (i.e., the ring contains forward pointers), the element block nearest the head when moving toward the head will in general be different from the element block nearest the head when moving away. This will not affect the validity of the expression we will derive, however.

Since, when $\Delta_i = 1$, we assume the duplicated head and its associated element block function as a single block, we define $e_i^o$ to be 0 when $\Delta_i = 1$. This leaves us with two possibilities: either the $a_i$-rings of the SSM are explicit rings with forward pointers (if $\phi_i = 1$) or the element blocks are stacked upon the head (if $\phi_i = 0$). In the first case $e_i^o$ will clearly be equal to $f_i$ and in the second case to $s_i$. Recalling that $s_i \leq f_i \leq h_i$, we see that even if $\phi_i' = 1$, choosing to follow a forward pointer or to step through the stack, as the case may be, is still preferable to following the head pointer.

Thus, $e_i^o$ will be given by the following expressions:

$$e_i^o = \overline{\Delta}_i \, (\phi_i f_i + \overline{\phi}_i \, s_i) \quad \text{for} \quad i \epsilon \{1, 2, \cdots, 10\}$$

Finally, let $e_i^*$ represent the number of time units required to move from an arbitrary element block of an $a_i$-ring of the SSM to the head of that ring, where $i \epsilon \{1, 2, \cdots 10\}$.

We should discuss for a moment what we mean by "an arbitrary element block". In tracing through a storage structure we may enter a ring via any of its element blocks and desire to move to its head. If there are k element blocks in the ring and if we are given no a priori information (which we assume we are not), then we are equally likely to enter the ring at any of the k element blocks. That is, the probability that we enter the ring via the j-th element block is 1/k for all $j \epsilon \{1, 2, \cdots, k\}$, where we assume that the element blocks of the ring are numbered consecutively from 1 to k in the order in which they appear in the ring. The number of the element block via which we expect to enter the ring is then given by the expectation of j, which we denote E[ j] .

$$\begin{aligned} E[\,j] &= \sum_{j=1}^{k} j\ (\frac{1}{k}) \\ &= \frac{1}{k} \sum_{j=1}^{k} j \\ &= \frac{1}{k}\ \frac{k(k+1)}{2} \\ &= \frac{k+1}{2} \end{aligned}$$

As a matter of notational convenience let us define

$$\hat{k} = \frac{k+1}{2}$$

When we refer to "an arbitrary element block", it is this block - the $\hat{k}$-th element block - to which we refer.

Let us now consider the amount of time required to move from the $\hat{k}_i$-th element block of an $a_i$-ring of the SSM to the head of that ring, where $i \epsilon \{1, 2, \cdots, 10\}$.

As is the case with $e^o_i$, if $\Delta_i = 1$, we define $e^*_i$ to be 0.

If $\phi'_i = 1$, very clearly we need only follow the head pointer in the element block to reach the head. Then $e^*_i = h_i$.

On the other hand, if $\phi'_i = 0$, we must move from the $\hat{k}_i$-th element block through all intervening element blocks in order to reach the head. We know, however, that moving from the $\hat{k}_i$-th block to the $\hat{k}_i$-1-st block requires $e_i$ units of time, as does moving from the $\hat{k}_i$-1-st block to the $\hat{k}_i$-2-nd block, and so on through the 2-nd block to the 1-st block. Similarly, we know that moving from the 1-st block (the block nearest the head) to the head requires $e^o_i$ units of time. Thus, a total of

$$e^o_i + (\hat{k}_i - 1)\, e_i$$

units of time are required to move from the $\hat{k}_i$-th element block to the head.

Therefore, $e^*_i$ may be characterized by the following expression:

$$e^*_i = \bar{\Delta}_i \{ \phi'_i h_i + \bar{\phi}'_i [ e^o_i + (\hat{k}_i - 1) e'_i ] \}$$

for $i \epsilon \{1, 2, \cdots 10\}$.

## 4.2.3 Sub-procedures and Their Time Costs

Upon closer inspection of our definitions for the primitive operations and their methods of implementation, we see that each method consists of a number of sub-procedures, many of which are common to several different methods.

For instance, one of these sub-procedures might be that sequence of steps required to search the SSM for all occurences of a particular relation symbol associated with a given source. Another might be that sequence of steps required to access all targets associated with a given source/relation symbol pair.

Since sub-procedures such as these appear several times in the definitions of the various methods and since our ultimate goal is the determination of the time cost for each method, it will be advantageous to develop time costs for the sub-procedures.

If we examine in detail the procedures used to implement the methods for the various primitive operations (as given in Table 4-2 and Appendix A), we see that they involve twenty different sub-procedures in two classes. The first class contains ten sub-procedures of the form "Determine all $x_1$ associated with $x_2$", and the second class contains ten sub-procedures of the form "Search for a particular $x_1$ associated with $x_2$". Let $\alpha_i$ denote the i-th sub-procedure in the first class and let $\alpha_i^*$ denote the i-th sub-procedure in the second class, where $i \in \{1, 2, \cdots 10\}$.

Using the terminology of the SSM (where we refer to particular blocks) as opposed to the terminology of the DSM (where we refer to sources, targets, and relation symbols), let us define first the sub-procedures $\alpha_i$ for $i \in \{1, 2, \cdots, 10\}$.

Let $\alpha_1$ represent the sequence of steps required to access all $b_1$-blocks associated with a given $b_7$-block. Similarly, let $\alpha_2$ represent the sequence of steps required to access all $b_1$-blocks associated with a given $b_6$-block. We note that the effects of $\alpha_1$ and $\alpha_2$ are identical. That is, the same $b_1$-blocks will be accessed by both $\alpha_1$ and $\alpha_2$ (assuming of course, that the given $b_6$-block and the given $b_7$-block are associated with one another). The time costs of these two sub-procedures will in general be different, however.

Let $\alpha_3$ represent the sequence of steps required to access all $b_{11}$-blocks associated with a given $b_5$-block, and let $\alpha_4$ represent the sequence of steps required to access all $b_{11}$-blocks associated with a given $b_6$-block. Clearly, $\alpha_3$ and $\alpha_4$ are the direct analogies of $\alpha_1$ and $\alpha_2$, respectively, for tracing a storage structure in the other direction.

Let $\alpha_5$ and $\alpha_6$ represent those sequences of steps required to access all $b_1$-blocks and all $b_{11}$-blocks, respectively, associated with all $b_6$-blocks which represent the same relation symbol.

Let $\alpha_7$ and $\alpha_8$ represent those sequences of steps required to access all $b_5$-blocks associated with the $b_1$-block(s) representing a given source and to access all $b_7$-blocks associated with the $b_{11}$-block(s) representing a given target, respectively.

Finally, let $\alpha_9$ and $\alpha_{10}$ represent those sequences of steps required to access all $b_{11}$-blocks associated with the $b_1$-block(s) representing a given source and to access all $b_1$-blocks associated with the $b_{11}$-block(s) representing a given target, respectively.

A summary of these sub-procedure descriptions appears in Table 4-3.

Consider now the sub-procedures $\alpha_i^*$ for $i \epsilon \{1, 2, \cdots 10\}$. Instead of accessing all of the blocks of a given type which are associated with some block or blocks, we may wish to access only enough of these blocks to find a particular one. If there are k blocks of a given type associated with some block of another type, we would expect that in order to access one particular block of the k blocks we must access on the average $\hat{k}$ blocks, the last one accessed being the block sought. An example of this might be looking for the $b_5$-block which represents a particular relation symbol and which is associated with a particular $b_1$-block.

Clearly, we can establish a one-to-one correspondence between the sub-procedures $\alpha_i$ and the sub-procedures $\alpha_i^*$. Thus, by substituting the phrase "a particular" for the word "all", we

Sub-procedure $\alpha$ represents the sequence of steps required to access all $x_1$ associated with $x_2$.

| $\alpha$ | $x_1$ | $x_2$ |
|---|---|---|
| $\alpha_1$ | $b_1$-blocks | a given $b_7$-block |
| $\alpha_2$ | $b_1$-blocks | a given $b_6$-block |
| $\alpha_3$ | $b_{11}$-blocks | a given $b_5$-block |
| $\alpha_4$ | $b_{11}$-blocks | a given $b_6$-block |
| $\alpha_5$ | $b_1$-blocks | all $b_6$-blocks which represent the same relation symbol |
| $\alpha_6$ | $b_{11}$-blocks | all $b_6$-blocks which represent the same relation symbol |
| $\alpha_7$ | $b_5$-blocks | the $b_1$ block(s) which represent a given source |
| $\alpha_8$ | $b_7$-blocks | the $b_{11}$-block(s) which represent a given target |
| $\alpha_9$ | $b_{11}$-blocks | the $b_1$-block(s) which represent a given source |
| $\alpha_{10}$ | $b_1$-blocks | the $b_{11}$-block(s) which represent a given target |

Table 4-3. Summary of Sub-Procedure Descriptions

may define sub-procedure $\alpha_i^*$ with the same statement used to define sub-procedure $\alpha_i$ for $i \in \{1,2,\cdots,10\}$. Also, Table 4-3 may be considered a summary of the sub-procedures $\alpha_i^*$ if "all" is replaced by "a particular" in the statement "Sub-procedure $\alpha$ represents the sequence of steps required to access all $x_1$ associated with $x_2$."

Let the functions $z_i(t)$ and $z_i^*(t)$ represent the number of time units required to perform $\alpha_i$ and $\alpha_i^*$, respectively, upon the storage structure represented by the SSM, where $i \in \{1,2,\cdots,10\}$. The variable $t$, which is used as an argument to the functions $z_i$ and $z_i^*$, represents the number of time units required to perform some operation upon each of the blocks accessed by $\alpha_i$ and $\alpha_i^*$, as the case may be.

The operations to which t refers may be divided into two classes: the first class contains those operations which compare the contents of some field of a block with some given quantity, indicating whether or not a match is made and the second class contains those operations which fetch (and record or display) the contents of some field of a block.

Before considering the operations in each of these two classes. let us define a number of basic quantities much as we did for $f_i$, $s_i$, and $h_i$. Let $f_a$ represent the number of time units required to follow a description block indicator from a $b_1$-block of the SSM to the corresponding description block. ($f_a$ is of importance only if $\sigma_1 = 1$, of course.) Similarly, let $f_p$ represent the number of time units required to follow a description block indicator from a $b_{11}$-block of the SSM to the corresponding description block.

Let $F_a$, $F_r$, and $F_p$ represent the number of time units required to follow a pointer in a source ring, a relation ring, and a target ring, respectively.

Let $c_d$ represent the number of time units required to compare a data item description (as it appears in a description block) with a known or given quantity. Similarly, let $c_r$ represent the number of time units required to compare a relation symbol name with a given quantity.

Finally, let $v_d$ and $v_r$ represent the number of time units required to fetch a data item description and a relation symbol name, respectively.

Return now to consideration of the two classes of operations. In the first class we will define five operations which will require the following respective numbers of time units to perform: $C_a$, $C_{r_1}$, $C_r$, $C_{r_2}$ and $C_p$. In the second class we will also define five operations which will require $V_a$, $V_{r_1}$, $V_r$, $V_{r_2}$, and $V_p$ units of time to perform, respectively.

$C_a$ and $V_a$ represent the number of time units required to compare against a given quantity and to fetch, respectively, the data item description associated with a given $b_1$-block of the SSM. If $\sigma_1 = 0$, the description block associated with a given $b_1$-block is attached directly to the $b_1$-block. In this case $C_a = c_d$ and $V_a = v_d$. On the other hand if $\sigma_1 = 1$, we must follow a description block indicator in order to reach the description block before we can compare or record the description. Thus, in this case $C_a = f_a + c_d$ and $V_a = f_a + v_d$. We may then characterize $C_a$ and $V_a$ by the following expressions:

$$C_a = \sigma_1 f_a + c_d$$

$$V_a = \sigma_1 f_a + v_d$$

$C_p$ and $V_p$ represent quantities analogous to $C_a$ and $V_a$ for the $b_{11}$-blocks of the SSM. It follows that

$$C_p = \sigma_2 f_p + c_d$$

$$V_p = \sigma_2 f_p + v_d$$

$C_r$ and $V_r$ represent the number of time units required to compare against a given quantity and to fetch, respectively, the relation symbol name associated with a given $b_6$-block of the SSM. If $\rho_2 = 1$, the $b_6$-block contains a relation symbol name field so that $C_r = c_r$ and $V_r = v_r$. If on the other hand $\rho_2 = 0$, we have two choices: (1) if $\rho_1$ or $\rho_3$ is 1, go to one of the $b_5$-blocks or $b_7$-blocks associated with the $b_6$-blocks and containing a relation symbol name field, or (2) go to the head of the relation ring passing through the $b_6$-block (which we assume contains a relation symbol name field). It so happens that $C_r$ and $V_r$ will be used only when $\rho_1 = 0$ or $\rho_3 = 0$, however. **In particular**, they are used only in conjunction with the sub-procedures $\alpha_7$, $\alpha_8$, $\alpha^*_7$, $\alpha^*_8$, which are used to move toward the "center" of the structure in search of one or more relation symbols. These sub-procedures first encounter either the $b_5$-blocks or the $b_7$-blocks. If these first blocks encountered do not contain a relation symbol name field, then the corresponding $b_6$-block must be accessed, which brings us to the point of this discussion. Let us assume that $\rho_1$ and $\rho_3$ must be equal when $\rho_2 = 0$. This implies that if only one type of block (of the $b_5$-blocks, $b_6$-blocks, and $b_7$- blocks) contains a relation symbol name field, then the $b_6$-blocks must contain that field (i.e., then $\rho_1 = 0$, $\rho_2 = 1$, and $\rho_3 = 0$). This is a rather arbitrary assumption which we motivate simply on the grounds of convenience in deriving the related expressions. It does, however, have a certain intuitive appeal in that it preserves some of the symmetry of the model. This assumption, coupled with the fact that $C_r$ and $V_r$ will used only when $\rho_1 = 0$ or $\rho_3 = 0$, rules out the first alternative above.

To reach the head of a relation ring, as suggested by the second alternative, we must follow $\hat{m}_r$ pointers. (Since there are on the average $m_r$ $b_6$-blocks which represent the same relation symbol, it follows that there are $m_r$ $b_6$-blocks in a given relation ring. We assume that we enter this ring via an arbitrary $b_6$-block in the ring. Hence, we must follow $\hat{m}_r$ pointers to reach the head.) It follows that $C_r = \hat{m}_r F_r + c_r$ and $V_r = \hat{m}_r F_r + v_r$. We may then characterize $C_r$ and $V_r$ by the following expressions:

$$C_r = \bar{\rho}_2 \hat{m}_r F_r + c_r$$

$$V_r = \bar{\rho}_2 \hat{m}_r F_r + v_r$$

$C_{r_1}$ and $V_{r_1}$ represent the number of time units required to compare against a given quantity and to fetch, respectively, the relation symbol name associated with a given $b_5$-block of the SSM. If $\rho_1 = 1$, clearly $C_{r_1} = c_r$ and $V_{r_1} = v_r$. If $\rho_1 = 0$, however, we will move to the $b_6$-block associated with the given $b_5$-block and apply the operations which give rise to $C_r$ and $V_r$. We know that the amount of time required to move from a $b_5$-block to the corresponding $b_6$-block is given by $e_5^*$. Therefore, in this case $C_{r_1} = e_5^* + C_r$ and $V_{r_1} = e_5^* + V_r$. $C_{r_1}$ and $V_{r_1}$ are then described by the following expressions:

$$C_{r_1} = \rho_1 c_r + \bar{\rho}_1 (e_5^* + C_r)$$

$$V_{r_1} = \rho_1 v_r + \bar{\rho}_1 (e_5^* + V_r)$$

$C_{r_2}$ and $V_{r_2}$ represent quantities analogous to $C_{r_1}$ and $V_{r_1}$ for the $b_7$-blocks of the SSM. It follows that

$$C_{r_2} = \rho_3 c_r + \bar{\bar{\rho}}_3 (e_6^* + C_r)$$

$$V_{r_2} = \rho_3 v_r + \bar{\rho}_3 (e_6^* + V_r)$$

It should be clear that only certain of the quantities $C_a$, $C_{r_1}$, $C_r$, $C_{r_2}$, $C_p$, $V_a$, $V_{r_1}$, $V_r$, $V_{r_2}$, and $V_p$ may be substituted for t in $z_i(t)$ and $z_i^*(t)$ for a given value of $i \in \{1, 2, \cdots, 10\}$. For example, only $C_a$ and $V_a$ apply. when $i = 1$.

Let us continue now with our consideration of the functions $z_i(t)$ and $z_i^*(t)$ by intially considering $z_i(t)$ for $i \in \{1, 2, \cdots, 10\}$.

Consider first $z_1(t)$, which represents the number of time units required to access all $b_1$-blocks associated with a given $b_7$-block. If we wish to compare with a given quantity the data item description associated with each $b_1$-block accessed, then $t = C_a$. On the other hand, if we wish to fetch the data item description associated with each $b_1$-block accessed, then $t = V_a$. In either event the steps required to access the $b_1$-blocks will be the same.

We may outline these steps as follows:

(1) Move to the $b_6$-block associated with the given $b_7$-block.

(2) Access all $b_5$-blocks associated with that $b_6$-block.

(3) Move to the $b_4$-block associated with each $b_5$-block.

(4) Access all $b_3$-blocks associated with each $b_4$-block.

(5) Move to the $b_2$-block associated with each $b_3$-block.

(6) Access all $b_1$-blocks associated with each $b_2$-block.

Of course, depending upon what transformations have been applied to the SSM, some of these steps may not have to be performed (i.e., the time required to perform the steps may be zero).

Let us examine the number of time units required to perform each of these six steps. Step (1) will clearly require $e^*_6$ units of time to perform.

To perform step (2) we must move from the $b_6$-block to the first $b_5$-block of the $a_5$-ring and then move sequentially through the remaining $K_{33}$-1 $b_5$-blocks of the ring. This will require $e^0_5 + K_{33}\, e'_5$ units of time. We have assumed that the last element block of the $a_5$-ring is treated exactly like the other element blocks in that we try to access the element block following it. Of course, there is no such block, but we assume the time required to determine this fact is given by $e'_5$. Hence, we multiply $e'_5$ by $K_{33}$ instead of by $K_{33}$-1.

The performance of step (3) normally requires $e^*_4$ units of time and since it must be performed once for every $b_5$-block associated with the given $b_7$-block, the total time required is $K_{33}\ e^*_4$ units.

Step (4) is similar to step (2) and we determine that it requires $e^o_3 + K_{22}\ e'_3$ units of time. Since it also must be performed once for every $b_5$-block, the total time required is $K_{33}(e^o_3 + K_{22}e'_3)$.

Note, however, that if $\phi_3 = 0$, $\phi_4 = 0$, $\phi_5 = 0$, and $\Delta_5 = 0$ ($\Delta_4 = 1$ in this case), then steps (3) and (4) are included in step (2). If both $\Delta_3 = 1$ and $\Delta_4 = 1$, this is only of academic interest since $e^*_4 = 0$, $e^o_3 = 0$, and $e'_3 = 0$ anyway, but if $\Delta_3 = 0$, it is of some importance. Given that $\phi_3 = 0$, $\phi_4 = 0$, $\phi_5 = 0$, $\Delta_3 = 0$, $\Delta_4 = 1$, and $\Delta_5 = 0$, we know that $b_3$-blocks are stacked upon their respective $b_4$-blocks, $b_4$-blocks are duplicated upon their associated $b_5$-blocks, and $b_5$-blocks are in turn stacked upon their respective $b_6$-blocks. Thus, intervening between each pair of $b_5$-blocks in a stack is a stack of $b_3$-blocks (which may, depending upon the values assigned to other decision variables, have stacks of blocks between pairs of them). It follows that in moving from one $b_5$-block in a stack to the succeeding $b_5$-block, we must step through each of the intervening $b_3$-blocks. (Because we are moving away from the heads of the stacks, we may not take advantage of any head pointers, of course.) Clearly then, step (2) encompasses steps (3) and (4) and the time costs of steps (3) and (4) may be disregarded. This also justifies our assumption that $e_5'$ represents the time required to determine that the last $b_5$-block in a given $a_5$-ring is indeed the last.

Step (5) requires $e_2^*$ units of time and must be performed once for every $b_3$-block associated with the given $b_7$-block, or $K_{32}$ times. Thus, its total required time is $K_{32}\, e_2^*$ units.

Step (6) is similar to steps (2) and (4). Its total required time is $K_{32}\,(e_1^0 + K_{11}\, e_1')$.

Note that if $\phi_1 = 0$, $\phi_2 = 0$, $\phi_3 = 0$, and $\Delta_3 = 0$, then steps (5) and (6) are included in step (4). This situation parallels that for the inclusion of steps (3) and (4) in step (2) exactly. By extending our reasoning, we can in fact show that if steps (5) and (6) are included in step (4) and if step (4) is in turn included in step (2), then steps (5) and (6) are included in step (2).

Finally, for each $b_1$-block accessed we perform that operation which is characterized by t.

Before summarizing $z_1(t)$ to this point, let us define a binary-valued <u>inclusion variable</u> $\lambda_i$ as follows:

$$\lambda_i = \phi_{i-1} + \phi_i + \phi_{i+1} + \Delta_{i+1} \quad \text{for } i \in \{2,4,6,8\}$$

$$\lambda_i = \phi_{i-1} + \phi_i + \phi_{i+1} + \Delta_{i-1} \quad \text{for } i \in \{3,5,7,9\}$$

(Recall our assumption that "+" indicates disjunction when applied to decision variables.)

We may now write the following expression for $z_1(t)$:

$$\begin{aligned} z_1(t) &= e^*_6 + e^o_5 + K_{33}e'_5 \\ &+ \lambda_4[K_{33}e^*_4 + K_{33}(e^o_3 + K_{22}e'_3)] \\ &+ \lambda_4[K_{32}e^*_2 + K_{32}(e^o_1 + K_{11}e'_1)] \\ &+ K_{31}\ t \end{aligned}$$

If we rearrange the terms of this expression somewhat, we can obtain the expression

$$\begin{aligned} z_1(t) = e^*_6 + e^o_5 + K_{33}\ [\quad & e'_5 + \lambda_4(e^*_4 + e^o_3)] \\ + K_{32}\ [\ & \lambda_4 e'_3 + \lambda_2\ (e^*_2 + e^o_1)] \\ + K_{31}\ [\ & \lambda_2 e'_1 + t] \end{aligned}$$

or alternatively

$$\begin{aligned} z_1(t) = e^*_6 + e^o_5 + \quad k_5\ [\quad & e'_5 + \lambda_4(e^*_4 + e^o_3)] \\ + \quad k_3k_5\ [\ & \lambda_4 e'_3 + \lambda_2(e^*_2 + e^o_1)] \\ + k_1k_3k_5[\ & \lambda_2 e'_1 + t] \end{aligned}$$

This expression is not quite complete, however, for we have ignored so far quantities such as the number of time units required to move from a pointer field for one ring to a pointer field for another ring passing through the same block (when such a situation exists).

We assume that when $\alpha_1$ is to be performed, the position indicator is pointing to the relation symbol name field of a given $b_7$-block. (If the $b_7$-blocks do not contain a relation symbol name field, $\alpha_1$ will not be used.) If $\phi_6 = 1$ or $\phi'_6 = 1$, we must step from the relation symbol name field to either the head pointer field or the forward pointer field of the given $b_7$-block in order to allow access to the corresponding $b_6$-block. As we discussed earlier, this will require $s_0$ units of time. Suppose instead that $\Delta_6 = 1$ and $\phi_5 = 1$. In this case we must step from the relation symbol name field of the $b_7$-block to the $a_5$-ring forward pointer field of the corresponding $b_6$-block. (Recall that $e_6$ is defined to be zero for $\Delta_6 = 1$ ) This will also require $s_0$ units of time. To carry this one more step, suppose $\Delta_6 = 1$, $\Delta_5 = 1$, and $\phi_4 = 1$ or $\phi'_4 = 1$. Here we must step from the relation symbol name field of the $b_7$-block to either the $a_4$-ring head pointer field or the $a_4$-ring forward pointer field of the corresponding $b_5$-block. There is, of course, only one $b_5$-block associated with the given $b_7$-block in this case. Also, $e^*_6 = 0$, $e^0_5 = 0$, and $e'_5 = 0$. As before, this move requires $s_0$ time units. Continuing our example in this manner we reach the point where $\Delta_6 = 1$, $\Delta_5 = 1$, $\Delta_4 = 1$, $\Delta_3 = 1$, $\Delta_2 = 1$, and $\Delta_1 = 1$. Clearly, there is exactly one $b_1$-block associated with the given $b_7$-block. Furthermore, $e^*_6 = 0$, $e^o_5 = 0$, $e'_5 = 0$, $e^*_4 = 0$, $e^o_3 = 0$, $e'_3 = 0$, $e^*_2 = 0$, $e^o_1 = 0$, and $e'_1 = 0$. We must in this case step from the relation symbol name field of the $b_7$-block to either the description block indicator or the description block itself, as the case may be. This also requires $s_o$ time units. Thus, whenever the following expression has a value of 1, we must step from the relation symbol name field of the given $b_7$-block to some other field in the structure a move which requires $s_o$ time units:

$$(\phi_6 \vee \phi'_6) \vee \Delta_6\phi_5 \vee \Delta_6\Delta_5(\phi_4 \vee \phi'_4) \vee \Delta_6\Delta_5\Delta_4\phi_3$$
$$\vee\ \Delta_6\Delta_5\Delta_4\Delta_3(\phi_2 \vee \phi'_2) \vee \Delta_6\Delta_5\Delta_4\Delta_3\Delta_2\phi_1$$
$$\vee\ \Delta_6\Delta_5\Delta_4\Delta_3\Delta_2\Delta_1$$

(We have used "V" instead of "+" to denote disjunction here because we will later want to arithmetically sum the values of expressions such as this.)

Suppose now that $\phi_6 = 1$ and $\phi_5 = 1$. If $\phi'_6 = 0$, we must follow the $a_6$-ring forward pointers to reach the $b_6$-block associated with the given $b_7$-block. In addition we are to follow the $a_5$-ring forward pointers to access all $b_5$-blocks associated with that $b_6$-block. Hence, at the $b_6$-block we must step from the $a_6$-ring forward pointer field to the $a_5$-ring forward pointer field. This move, as we know, requires

$s_o$ time units. If $\phi'_6 = 1$, we can follow the $a_6$-ring head pointer directly to the $a_5$-ring forward pointer without incurring the additional $s_o$ units of time. Suppose next that $\phi_6 = 1$, $\phi'_6 = 0$, $\Delta_5 = 1$, and $\phi_4 = 1$ or $\phi'_4 = 1$. In this case we must step from the $a_6$-ring forward pointer to either the $a_4$-ring head pointer or the $a_4$-ring forward pointer. Again, $s_o$ units of time are required. If we continue with this line of reasoning, we can obtain another expression

$$\phi_6\overline{\phi}'_6\phi_5 \vee \phi_6\overline{\phi}'_6\Delta_5(\phi_4 \vee \phi'_4) \vee \phi_6\overline{\phi}'_6\Delta_5\Delta_4\phi_3$$

$$\vee\ \phi_6\overline{\phi}'_6\Delta_5\Delta_4\Delta_3(\phi_2 \vee \phi'_2) \vee \phi_6\overline{\phi}'_6\Delta_5\Delta_4\Delta_3\Delta_2\phi_1$$

$$\vee\ \phi_6\overline{\phi}'_6\Delta_5\Delta_4\Delta_3\Delta_2\Delta_1$$

which, when its value is 1, implies a step requiring $s_o$ units of time.

Using the same type of reasoning for cases for which $\phi_5 = 1$, we can develop the expression

$$\phi_5(\phi_4 \vee \phi'_4) \vee \phi_5\Delta_4\phi_3 \vee \phi_5\Delta_4\Delta_3 \ (\phi_2 \vee \phi'_2)$$

$$\vee\ \phi_5\Delta_4\Delta_3\Delta_2\phi_1 \vee \phi_5\Delta_4\Delta_3\Delta_2\Delta_1$$

In this case, however, when the value of the expression is 1, $k_5$ steps of $s_o$ time units are required. This follows, of course, from the fact that the step must be performed for each of the $k_5$ $b_5$-blocks associated with the given $b_7$-block.

If we carry on in this manner, we can develop several more expressions of this type. Let us consolidate all of these expressions and their respective multipliers into a single expression which we will designate $z_1^0$ as follows:

$$z_1^0 = \{((\phi_6 \vee \phi_6')\ \Delta_6\phi_5 \vee \Delta_6\Delta_5(\phi_4 \vee \phi_4') \vee \cdots \vee \Delta_6\Delta_5\Delta_4\Delta_3\Delta_2\Delta_1)$$

$$+(\phi_6\overline{\phi}'_6\phi_5 \vee \phi_6\overline{\phi}'_6\Delta_5(\phi_4 \vee \phi_4') \vee \phi_6\overline{\phi}'_6\Delta_5\Delta_4\phi_3 \vee \cdots \vee \phi_6\overline{\phi}'_6\Delta_5\Delta_4\Delta_3\Delta_2\Delta_1)$$

$$+k_5[(\phi_5(\phi_4 \vee \phi_4') \vee \phi_5\Delta_4\phi_3 \vee \phi_5\Delta_4\Delta_3(\phi_2 \vee \phi_2') \vee \cdots \vee \phi_5\Delta_4\Delta_3\Delta_2\Delta_1)$$

$$+(\phi_4\overline{\phi}'_4\phi_3 \vee \phi_4\overline{\phi}'_4\Delta_3(\phi_2 \vee \phi_2') \vee \phi_4\overline{\phi}'_4\Delta_3\Delta_2\phi_1 \vee \phi_4\overline{\phi}'_4\Delta_3\Delta_2\Delta_1)]$$

$$+k_3k_5[(\phi_3(\phi_2 \vee \phi_2') \vee \phi_3\Delta_2\phi_1 \vee \phi_3\Delta_2\Delta_1)$$

$$+(\phi_2\overline{\phi}'_2\phi_1 \vee \phi_2\overline{\phi}'_2\Delta_1)]$$

$$+k_1k_3k_5[\ \phi_1]\}\ s_0$$

In this expression "+" indicates arithmetic summation.

$z_1^0$ indicates the number of time units by which our current expression for $z_1(t)$ is deficient. The situation is easily remedied:

$$z_1(t) = e^*_6 + e^O_5 + k_5[\; e'_5 + \lambda_4(e^{[illegible]}\; e^O_3)]$$

$$+ k_3k_5[\lambda_4 e'_3 + \lambda_2(e^{[illegible]} + e^O_1)]$$

$$+ k_1k_3k_5[\lambda_2 e'_1 + t [illegible] + z^O_1$$

Consider next $z_2(\;)$, which represents the number of time units required to access all $b_1$-blocks associated with a given $b_6$-block. The derivation of an expression for $z_2(t)$ follows almost exactly the derivation of the expression for $z_1(t)$. The only difference is that for $z_2(t)$ there is no need (obviously) to move from a $b_7$-block to the $b_6$-block. Thus,

$$z^O_2 = \{(\phi_5 \vee \Delta_5(\phi_4 \vee \phi'_4) \vee \Delta_5\Delta_4\phi_3 \vee \Delta_5\Delta_4\Delta_3(\phi_2 \vee \phi'_2) \vee \cdots \vee \Delta_5\Delta_4\Delta_3\Delta_2\Delta_1)$$

$$+k_5[(\phi_5(\phi_4 \vee \phi'_4) \vee \phi_5\Delta_4\phi_3 \vee \phi_5\Delta_4\Delta_3(\phi_2 \vee \phi'_2) \vee \cdots \vee \phi_5\Delta_4\Delta_3\Delta_2\Delta_1)$$

$$+(\phi_4\bar{\phi}'_4\phi_3 \vee \phi_4\bar{\phi}'_4\Delta_3(\phi_2 \vee \phi'_2) \vee \phi_4\bar{\phi}'_4\Delta_3\Delta_2\phi_1 \vee \phi_4\bar{\phi}'_4\Delta_3\Delta_2\Delta_1)]$$

$$+k_3k_5[(\phi_3(\phi_2 \vee \phi'_2) \vee \phi_3\Delta_2\phi_1 \vee \phi_3\Delta_2\Delta_1)$$

$$+(\phi_2\bar{\phi}'_2\phi_1 \vee \phi_2\bar{\phi}'_2\Delta_1)]$$

$$+k_1k_3k_5[\phi_1]\}\; s_O$$

and

$$\begin{aligned} z_2(t) = e^o_5 + \quad & k_5[\ e'_5 + \lambda_4(e^*_4 + e^o_3)] \\ + \ & k_3k_5[\lambda_4 e'_3 + \lambda_2(e^*_2 + e^o_1)] \\ + \ & k_1k_3k_5[\lambda_2 e'_1 + t] + z^o_2 \end{aligned}$$

In deriving $z^o_2$, we have assumed that the position indicator is intially pointing to the relation symbol name field of the given $b_6$-block. If $\rho_2 = 0$, however, we may assume that the indicator is pointing to the relation ring pointer field. (Recall that the relation ring is used to determine the relation symbol for a given $b_6$-block when $\rho_2 = 0$.) This does not affect the validity of our expression for $z^o_2$.

Since $\alpha_3$ and $\alpha_4$ are direct analogies of $\alpha_1$ and $\alpha_2$, respectively, it follows that $z_3(t)$ and $z_4(t)$ parallel exactly $z_1(t)$ and $z_2(t)$, respectively. The actual expressions for $z_3(t)$ and $z_4(t)$ may be found in Appendix C which contains a complete summary of $z_i(t)$ and $z^o_i$ or all $i \epsilon \{1, 2, \cdots, 10\}$.

Next consider the derivation of an expression for $z_5(t)$, which represents the number of time units required to access all $b_1$-blocks associated with all $b_6$-blocks which represent the same relation symbol. It should be clear that $\alpha_5$ can be performed by accessing all the $b_6$-block in a given relation ring and then applying $\alpha_2$ to each of those $b_6$-blocks.

Since there are $m_r$ $b_6$-blocks in a given relation ring, the number of time units required to access these $b_6$-blocks may be given by $(m_{r+1})\ F_r$. $m_rF_r$ time units are required to access the blocks, and $F_r$ time units are required to follow the pointer from the last $b_6$-block in the relation ring to the head, which indicates that all the $b_6$-blocks have been considered.

It follows that

$$z_5(t) = (m_{r+1})\ F_r + m_r z_2(t)$$

Although we have no explicit need for $z_5^o$, we know that $z_5^o = m_r\, z_2^o$.

$z_6(t)$ is analogous to $z_5(t)$ and, therefore, will not be considered in detail here. An expression for $z_6(t)$ appears in Appendix C.

Next in line for consideration is $z_7(t)$, which represents the number of time units required to access all $b_5$-blocks associated with the $b_1$-block(s) which represent a given course. By applying arguments similar to those used above, we can obtain the following expression:

$$\begin{aligned} z_7(t) = (m_a+1)\ F_a + m_a\ \{ & e_1^* + e_2^o \\ & + k_2[\ e_2' + \lambda_3(e_3^* + e_4^o)] \\ & + k_2k_4[\lambda_3 e_4' + t\ ]\} + z_7^o \end{aligned}$$

We have assumed here that a source ring is used to access the $b_1$-block or $b_1$-blocks representing a given source. If we should desire to assume that the locations of these $b_1$-blocks are known a priori (perhaps as the result of some other operation), we may set $F_a$ to zero and still use the given expression.

Derivation of an expression for $z_7^o$ proceeds along lines similar to the derivations for $z_1^o$, $z_2^o$, etc., but with some distinct differences. We assume that when we initially enter a $b_1$-block the position indicator points to the source ring pointer field of that block. Our immediate goal is to trace through the storage structure to access all $b_5$-blocks associated with the given $b_1$-block. For each $b_5$-block accessed, either we are to compare the relation symbol which it represents with some known quantity or we are to fetch that relation symbol. In either event we must determine what the relation symbol is. If $\rho_1 = 1$, we may determine what the relation symbol is simply by accessing the $b_5$-block. On the other hand if $\rho_1 = 0$, we must access the corresponding $b_6$-block. As a result, our expression for $z_7^o$ will appear as follows:

$$\begin{aligned} z_7^o = m_a \{ & ((\phi_1 \vee \phi_1') \vee \Delta_1 \phi_2 \vee \Delta_1 \Delta_2 (\phi_3 \vee \phi_3') \vee \Delta_1 \Delta_2 \Delta_3 \phi_4 \\ & \vee \Delta_1 \Delta_2 \Delta_3 \Delta_4 (\rho_1 \vee \phi_5 \vee \phi_5') \vee \Delta_1 \Delta_2 \Delta_3 \Delta_4 \Delta_5 \bar{\rho}_1) \\ & + (\phi_1 \bar{\phi}_1' \phi_2 \vee \phi_1 \bar{\phi}_1' \Delta_2 (\phi_3 \vee \phi_3') \vee \phi_1 \bar{\phi}_1' \Delta_2 \Delta_3 \phi_4 \\ & \vee \phi_1 \bar{\phi}_1' \Delta_2 \Delta_3 \Delta_4 (\rho_1 \vee \phi_5 \vee \phi_5') \vee \phi_1 \bar{\phi}_1' \Delta_2 \Delta_3 \Delta_4 \Delta_5 \bar{\rho}_1) \end{aligned}$$

$$+ k_2[ (\phi_2(\phi_3 \vee \phi'_3) \vee \phi_2\Delta_3\phi_4 \vee \phi_2\Delta_3\Delta_4(\rho_1 \vee \phi_5 \vee \phi'_5) \vee \phi_2\Delta_3\Delta_4\Delta_5\bar{\rho}_1)$$

$$+(\phi_3\bar{\phi}'_3\phi_4 \vee \phi_3\bar{\phi}'_3\Delta_4(\rho_1 \vee \phi_5 \vee \phi'_5) \vee \phi_3\bar{\phi}'_3\Delta_4\Delta_5\bar{\rho}_1)]$$

$$+k_2k_4 [ (\phi_4(\rho_1 \vee \phi_5 \vee \phi'_5) \vee \phi_4\Delta_5\bar{\rho}_1) + (\phi_5\bar{\phi}'_5\bar{\rho}_1)] \} s_o$$

To clarify the intent of this expression let us briefly examine the terms

$$\Delta_1\Delta_2\Delta_3\Delta_4(\rho_1 \vee \phi_5 \vee \phi'_5) \vee \Delta_1\Delta_2\Delta_3\Delta_4\Delta_5\bar{\rho}_1$$

which apply when $\Delta_1 = 1$, $\Delta_2 = 1$, $\Delta_3 = 1$, and $\Delta_4 = 1$. If $\rho_1 = 1$, we need only step from the source ring pointer field of the given $b_1$-block to the relation symbol name field of the associated $b_5$-block in order to be able to determine what the relation symbol is. On the other hand if $\rho_1 = 0$, either we must step to the $a_5$-ring forward pointer field or head pointer field of the $b_5$-block (if $\phi_5 = 1$ or $\phi'_5 = 1$) in order to access the corresponding $b_6$-block or we must step directly to the relation symbol name field or the relation ring pointer field of the $b_6$-block (if $\Delta_5 = 1$).

$z_8(t)$ is analogous to $z_7(t)$ and so will not be considered here but may be found in Appendix C.

$z_9(t)$, which represents the number of time units required to access all $b_{11}$-blocks associated with the $b_1$-block(s) which represent a given source, is similar in all respects to those $z_i(t)$ considered above. Without any difficulty ve can derive the following expression to represent it:

$$\begin{aligned}
z_9(t) &= (m_a+1)\bar{F}_a + m_a\{e^*_1 + e^o_2 \\
&+ k_2 \quad [\ e'_2 + \lambda_3(e^*_3+e^o_4)] \\
&+ k_2k_4 \quad [\lambda_3 e'_4 + \lambda_5(e^*_5 + e^o_6)] \\
&+ k_2k_4k_6 \quad [\lambda_5 e'_6 + \lambda_7(e^*_7+e^o_8)] \\
&+ k_2k_4k_6k_8[\lambda_7 e'_8 + \lambda_9(e^*_9 + e^o_{10})] \\
&+ k_2k_4k_6k_8k_{10}[\lambda_9 e'_{10} + t]\} + z^o_9
\end{aligned}$$

where

$$\begin{aligned}
z^o_9 = m_a\ \{&((\phi_1 \vee \phi'_1) \vee \Delta_1\phi_2\ '\Delta_1\Delta_2(\phi_3 \vee \phi'_3) \vee \cdots \vee \Delta_1\Delta_2\Delta_3\Delta_4\Delta_5\Delta_6\Delta_7\Delta_8\Delta_9\Delta_{10}) \\
&+(\phi_1\bar{\phi}'_1\phi_2 \vee \phi_1\bar{\phi}'_1\Delta_2(\phi_3 \vee \phi'_3) \vee \cdots \vee \phi_1\bar{\phi}'_1\Delta_2\Delta_3\Delta_4\Delta_5\Delta_6\Delta_7\Delta_8\Delta_9\Delta_{10})
\end{aligned}$$

$$\begin{aligned}
+K_{12}[&(\phi_2(\phi_3 \vee \phi'_3) \vee \phi_2\Delta_3\phi_4 \vee \phi_2\Delta_3\Delta_4(\phi_5 \vee \phi'_5) \vee \cdots \vee \phi_2\Delta_3\Delta_4\Delta_5\Delta_6\Delta_7\Delta_8\Delta_9\Delta_{10}) \\
&+(\phi_3\bar{\phi}'_3\phi_4 \vee \phi_3\bar{\phi}'_3\Delta_4(\phi_5 \vee \phi'_5) \vee \cdots \vee \phi_3\bar{\phi}'_3\Delta_4\Delta_5\Delta_6\Delta_7\Delta_8\Delta_9\Delta_{10})]
\end{aligned}$$

$$\begin{aligned}
+K_{13}[&(\phi_4(\phi_5 \vee \phi'_5) \vee \phi_4\Delta_5\phi_6 \vee \phi_4\Delta_5\Delta_6(\phi_7 \vee \phi'_7) \vee \cdots \vee \phi_4\Delta_5\Delta_6\Delta_7\Delta_8\Delta_9\Delta_{10}) \\
&+(\phi_5\bar{\phi}'_5\phi_6 \vee \phi_5\bar{\phi}'_5\Delta_6(\phi_7 \vee \phi'_7) \vee \cdots \vee \phi_5\bar{\phi}'_5\Delta_6\Delta_7\Delta_8\Delta_9\Delta_{10})]
\end{aligned}$$

$$+K_{14}[\,(\phi_6(\phi_7 \vee \phi'_7) \vee \phi_6\Delta_7\phi_8 \vee \phi_6\Delta_7\Delta_8(\phi_9 \vee \phi'_9) \vee \cdots \vee \phi_6\Delta_7\Delta_8\Delta_9\Delta_{10})$$

$$+(\phi_7\bar{\phi}'_7\phi_8 \vee \phi_7\bar{\phi}'_7\Delta_8(\phi_9 \vee \phi'_9) \vee \cdots \vee \phi_7\bar{\phi}'_7\Delta_8\Delta_9\Delta_{10})]$$

$$+K_{15}[\,(\phi_8(\phi_9 \vee \phi'_9) \vee \phi_8\Delta_9\phi_{10} \vee \phi_8\Delta_9\Delta_{10})$$

$$+(\phi_9\bar{\phi}'_9\phi_{10} \vee \phi_9\bar{\phi}'_9\Delta_{10})]$$

$$+K_{16}[\,\phi_{10}]\,\}\,s_o$$

Finally, $z_{10}(t)$, which is analogous to $z_9(t)$, appears in Appendix C. Let us now consider the functions $z^*_i(t)$ for $i \epsilon \{1, 2, \cdots 10\}$.

Consider the derivation of an expression for $z^*_1(t)$. There are $k_5$ $b_5$-blocks, $k_3k_5$ $b_3$-blocks, and $k_1k_3k_5$ $b_1$-blocks associated with a given $b_7$-block. (To simplify notation, we will use $K_{33}, K_{32}$, and $K_{31}$ in place of $k_5$, $k_3k_5$, and $k_1k_3k_5$, respectively) Because of the uniform distribution of blocks in the SSM, we can say in general that when we have accessed $\hat{K}_{31}$ $b_1$-blocks associated with a given b7-block, we will have accessed $\hat{K}_{32}$ $b_3$-blocks and $\hat{K}_{33}$ $b_5$-blocks associated with the $b_7$-block.

Since $\alpha^*_1$ and $\alpha_1$ are so closely related, one would expect a strong similarity between $z^*_1(t)$ and $z_1(t)$, and indeed there is. Using very much the same arguments we used for $z_1(t)$, we obtain the following expression for $z^*_1(t)$:

$$z_1^*(t) = e_6^* + e_5^0 + (\hat{K}_{33}-1) \quad e_5' + \hat{K}_{33}\lambda_4(e_4^* + e_3^0)$$
$$+ (\hat{K}_{32}-1)\lambda_4 e_3' + \hat{K}_{32}\lambda_2(e_2^* + e_1^0)$$
$$+ (\hat{K}_{31}-1)\lambda_2 e_1' + \hat{K}_{31}\, t + z_1^{0*}$$

where $z^{0*}_1$ performs the same function for $z_1^*(t)$ as $z_1^0$ does for $z_1(t)$.

It may be helpful to make a comment or two about this expression. Note that the coefficients of $e_5'$, $e_3'$, and $e_1'$ are of the form k-1, whereas in the expression for $z_1(t)$ these coefficients are of the form k (i.e., they are not decremented by 1). As we indicated in the derivation of the expression for $z_1(t)$, if we want to access all blocks of a given type, then even when we encounter the last block we must try to access one more in order to determine that all the blocks have been considered. If there are k such blocks, this means we must perform k access operations (not including the operation required to access the first block from the head), which accounts for coefficients of the form k in the expression for $z_1(t)$. If on the other hand we are looking for a particular block, then we may stop accessing blocks once we have encountered the one sought. This means that access operations need be performed only for those blocks which precede the block sought. Thus, if the block sought is the k-th block encountered (of a given type), we need perform only k-1 access operations, which accounts for coefficients of the form k-1 in the expression for $z_1^*(t)$.

Even though in this case we are seeking a particular $b_1$-block, this reasoning clearly applies to the $b_3$-blocks and the $b_5$-blocks at the intermediate levels, also. For instance, we can view this situation as searching for the particular $b_3$-block (associated with the given $b_7$-block, of course) with which the $b_1$-block sought is associated and searching for the particular $b_5$-block with which the $b_1$-block sought is associated.

At this point we bring up a note of caution: the expression for $z^*_1(t)$ above is correct as far as it goes, but it is incomplete.

Suppose for instance that $\lambda_4 = 0$, that is, $e^*_4$, $e^o_3$, and $e'_3$ are covered by $e'_5$. Of course, $e'_5$ applies only to the first $\hat{K}_{33}$-1 $b_5$-blocks encountered, not the last one.

This means that $e^*_4$ and $e^o_3$ are required once to account for the $b_3$-blocks associated with the last $b_5$-block encountered. $e'_3$ will also be required a number of times (to be determined) to account for these $b_3$-blocks. Normally (when $\lambda_4 = 1$) $e'_3$ is required for $\hat{K}_{32}$-1 $b_3$-blocks but $(\hat{K}_{33}-1)\ k_3$ occurrences of $e'_3$ are covered by $e'_5$ when $\lambda_4 = 0$. Therefore, we will require $(\hat{K}_{32}-1) - (\hat{K}_{33}-1)\ k_3$ occurrences of $e'_3$ when $\lambda_4 = 0$.

$$(\hat{K}_{32}-1) - (\hat{K}_{33}-1)\, k_3$$

$$= \left(\frac{k_3k_5+1}{2} - 1\right) - \left(\frac{k_5+1}{2} - 1\right) k_3$$

$$= \frac{k_3k_5-1}{2} - \frac{k_3k_5 - k_3}{2}$$

$$= \frac{k_3-1}{2}$$

$$= \hat{k}_3-1 = \hat{K}_{22}-1$$

Similarly, $e^*_2$ and $e^o_1$ are required once and $e'_1$ is required $\hat{K}_{11}-1$ times when $\lambda_2 = 0$.

Taking these factors into account, we obtain the following expression for $z^*_1(t)$:

$$\begin{aligned} z^*_1(t) &= e^*_6+e^o_5 + (\hat{K}_{33}-1)e'_5 + [\,\lambda_4(\hat{K}_{33}-1) + 1]\,(e^*_4 + e^o_3) \\ &+ (\lambda_4\hat{K}_{32}+\bar{\lambda}_4\hat{K}_{22}-1)\, e'_3 + [\,\lambda_2(\hat{K}_{32}-1) + 1]\,(e^*_2 + e^o_1) \\ &+ (\lambda_2\hat{K}_{31} + \bar{\lambda}_2\hat{K}_{11}-1)\, e'_1 + \hat{K}_{31}\, t + z^{o*}_1 \end{aligned}$$

The expression for $z^{o*}_1$ follows directly from the expression for $z^o_1$. All operations involved remain exactly the same; only the numbers of blocks are different.

$$z_1^{o*} = \{((\phi_6 \vee \phi_6') \vee \Delta_6 \phi_5 \vee \Delta_6 \Delta_5 (\phi_4 \vee \phi_4') \vee \cdots \vee \Delta_6 \Delta_5 \Delta_4 \Delta_3 \Delta_2 \Delta_1)$$

$$+ (\phi_6 \bar{\phi}_6' \phi_5 \vee \phi_6 \bar{\phi}_6' \Delta_5 \ (\phi_4 \vee \phi_4') \vee \phi_6 \bar{\phi}_6' \Delta_5 \Delta_4 \phi_3 \vee \cdots \vee \phi_6 \bar{\phi}_6' \ \Delta_5 \Delta_4 \Delta_3 \Delta_2 \Delta_1)$$

$$+\hat{K}_{33}[\ (\phi_5 (\phi_4 \vee \phi_4') \vee \phi_5 \Delta_4 \phi_3 \vee \phi_5 \Delta_4 \Delta_3 (\phi_2 \vee \phi_2') \vee \cdots \vee \phi_5 \Delta_4 \Delta_3 \Delta_2 \Delta_1)$$

$$+ (\phi_4 \bar{\phi}_4' \phi_3 \vee \phi_4 \bar{\phi}_4' \Delta_3 (\phi_2 \vee \phi_2') \vee \phi_4 \bar{\phi}_4' \Delta_3 \Delta_2 \phi_1 \vee \phi_4 \bar{\phi}_4' \Delta_3 \Delta_2 \Delta_1)]$$

$$+\hat{K}_{32}[\ (\phi_3 (\phi_2 \vee \phi_2') \vee \phi_3 \Delta_2 \phi_1 \vee \phi_3 \Delta_2 \Delta_1)$$

$$+ (\phi_2 \bar{\phi}_2' \phi_1 \vee \phi_2 \bar{\phi}_2' \Delta_1)]$$

$$+\hat{K}_{31}[\ \phi_1]\ \}\ s_o$$

The expressions for all other $z_i^*(t)$ and their corresponding $z_i^{o*}$ can be obtained as above and are summarized in Appendix C.

### 4.2.4 Primitive Operation Time Costs

We now have at our disposal the information required to return to our consideration of the time costs $t_{ij}$ associated with the methods for implementing the various primitive operations.

Let us consider the derivation of an expression for $t_{11}$, the number of time units required to perform $Q_1$ using its first method. Recall the basic steps required to implement this method:

(1) Find $d_i$ in the storage structure.

(2) Examine all relation symbols associated with $d_i$ in search of $r_j$.

(3) If $r_j$ is found, examine all targets in the corresponding target set in search of $p_k$.

Step (1) amounts to finding the head of the source ring containing those $b_1$-blocks which represent $d_j$. We will assume that finding the head of a source ring requires $T_a$ time units. (We can assume that this step is accomplished by using $d_i$ as a key to a dictionary, for instance.) Since we will have need of these quantities later, also assume that finding the head of a relation ring or a target ring requires $T_r$ and $T_p$ time units, respectively.

If we have assumed that $F_a = 0$ (i.e., that we do not use the source ring to access the $b_1$-blocks representing a given source, but rather we know a priori where these $b_1$-blocks are located), then we may assume that $T_a = 0$, also.

Now consider steps (2) and (3). Suppose that we do not find $r_j$ among the relation symbols associated with $d_i$. This means that we will have to examine all $b_5$-blocks associated with the $b_1$-block(s) representing $d_i$ in search of one which represents the relation symbol

$r_j$, but without success. Of course, this is simply sub-procedure $\alpha_7$, which has a time cost $z_7(C_{r_1})$.

Suppose next that we do find $r_j$ but not $p_k$. If there is but a single copy of each $b_5$-block associated with the $b_1$-blocks representing $d_i$ (i.e., if $m_{r_1} = 1$), the $b_5$-block representing $r_j$ may be found via $\alpha_7^*$, which has a time cost $z_7^*(C_{r_1})$. We will then examine via $\alpha_3$ all $b_{11}$-blocks associated with the $b_5$-block representing $r_j$ in search of $p_k$, which will not be found. The time cost required for this is $z_3(C_p)$. If on the other hand $m_{r_1} > 1$, we must examine all $b_5$-blocks to insure that we have found all those representing $r_j$. Then for each $b_5$-block representing $r_j$ we must perform $\alpha_3$. The time cost required in this case will be $z_7(C_{r_1}) + m_{r_1} z_3(C_p)$.

Finally, suppose that we find both $r_j$ and $p_k$. In this case we may stop accessing $b_5$-blocks as soon as $p_k$ is found. Thus, the time cost required to find $r_j$ will be $z_7^*(C_{r_1})$. Since there are $m_{r_1}$ $b_5$-blocks which represent $r_j$ (and which are associated with the $b_1$-block(s) representing $d_i$), we would expect $p_k$ to be associated with the $\hat{m}_{r_1}$-th $b_5$-block representing $r_j$. This means that for this $b_5$-block we need examine only part of the $b_{11}$-blocks associated therewith at a time cost $z_3^*(C_p)$, but for the first $\hat{m}_{r_1}-1$ $b_5$-blocks representing $r_j$ we must examine all associated $b_{11}$-blocks at a time cost $z_3(C_p)$ each.

Implicit in this discussion has been the assumption that $\rho_1 = 1$. If in fact $\rho_1 = 0$, we may use $\alpha_4$ and $\alpha^*_4$ in place of $\alpha_3$ and $\alpha^*_3$, respectively. The reason for this, of course, is that the operation characterized by $C_{r_1}$ must access the $b_6$-block associated with a given $b_5$-block when $\rho_1 = 0$, and we may use this $b_6$-block instead of the $b_5$-block as our starting point in the search for $p_k$. Clearly, $z_4(C_p)$ and $z^*_4(C_p)$ will then be used in place of $z_3(C_p)$ and $z^*_3(C_p)$ , respectively.

Let us now summarize the time cost for each of the three situations:

(1) $r_j$ is not found

$$z_7(C_{r_1})$$

(2) $r_j$ is found, but $p_k$ is not

$$(\bar{\Delta}_6+\bar{\Delta}_8)z^*_7(C_{r_1})+\Delta_6\Delta_8 z_7(C_{r_1}) + m_{r_1}\ [\rho_1 z_3(C_p)+\bar{\rho}_1 z_4(C_p)]$$

(3) both $r_j$ and $p_k$ are found

$$z^*_7(C_{r_1})+(\hat{m}_{r_1}-1)[\rho_1 z_3(C_p)+\bar{\rho}_1 z_4(C_p)] +[\rho_1 z^*_3(C_p)+\bar{\rho}_1+z^*_4(C_p)]$$

(Note that $\Delta_6 = 0$ or $\Delta_8 = 0$ implies $m_{r_1} = 1$ and $\Delta_6 = 1$ and $\Delta_8 = 1$ implies in general that $m_{r_1} > 1$.)

If we assume for the moment that the probability that the first of these situations occurs is given by $y_1$, the probability that the second occurs is given by $y_2$, and the probability that the third occurs is given by $y_3$, then the time cost $t_{11}$ will be given by the following expression:

$$
\begin{aligned}
t_{11} = T_a &+ y_1 z_7(C_{r_1}) \\
&+ y_2 \{(\bar{\Delta}_6+\bar{\Delta}_8)z^*_7(C_{r_1})+\Delta_6\Delta_8 z_7(C_{r_1})+m_{r_1}[\rho_1 z_3(C_p)+\bar{\rho}_1 z_4(C_p)]\} \\
&+ y_3\{z^*_7(C_{r_1})+(\hat{m}_{r_1}-1)[\rho_1 z_3(C_p)+\bar{\rho}_1 z_4(C_p)]+[\rho_1 z^*_3(C_p)+\bar{\rho}_1 z^*_4(C_p)]\}
\end{aligned}
$$

Let us now investigate the determination of the probabilities $y_1$, $y_2$, and $y_3$ (along with some other probabilities for which we will have use later).

Let us define an event A as "the relation symbol r is associated with the source d", where $r \epsilon P$ and $d \epsilon \Delta$. We can determine the probability that event A is true, $Pr\{A\}$, as follows.

Associated with each $d \epsilon \Delta$ are $k^o_2 k^o_4$ elements of $P$. Clearly, for any $d \epsilon \Delta$ there are then $k^o_2 k^o_4$ elements of the $k^o_r$ elements of $P$ for which A will be true. Thus,

$$
Pr\{A\} = \frac{k^o_2 k^o_4}{k^o_r}
$$

Similarly, associated with each $r \epsilon P$ are $m_r k^o_1 k^o_3$ elements of $\Delta$. For any $r \epsilon P$ there are then $m_r k^o_1 k^o_3$ elements of the $k^o_a$ elements of $\Delta$ for which A will be true. It follows that

$$\Pr\{A\} = \frac{m_r k_1^o k_3^o}{k_a^o}$$

Recall, however, that

$$m_r = \frac{k_2^o k_4^o k_a^o}{k_1^o k_3^o k_r^o}$$

Making this substitution for $m_r$ yields once again

$$\Pr\{A\} = \frac{k_2^o k_4^o}{k_r^o}$$

We can obtain this same result in a somewhat different way. Since $|\Delta| = k_a^o$ and $|P| = k_r^o$, there are $k_a^o k_r^o$ possible (d, r) - pairs. However, since there are $k_2^o k_4^o$ elements of P associated with each element of $\Delta$, only $k_2^o k_4^o k_a^o$ (d, r) - pairs actually cause event A to be true. Thus,

$$\Pr\{A\} = \frac{k_2^o k_4^o k_a^o}{k_a^o k_r^o} = \frac{k_2^o k_4^o}{k_r^o}$$

Let us define an event B as "the relation symbol r is associated with the target p", where $r \in P$ and $p \in \Pi$. In a manner similar to that for $\Pr\{A\}$ we can show that

$$\Pr\{B\} = \frac{k_7^o k_9^o}{m_{r_p} k^o_r}$$

For instance, associated with each $p \epsilon \Pi$ are $k_7^o k_9^o / m_{r_p}$ elements of P. Since $|P| = k_r^o$, the expression for $\Pr\{B\}$ follows directly.

Let A' be the event "the relation symbol r which a given $b_5$-block (or $b_6$-block) represents is associated with the source d", where $r \epsilon P$ and $d \epsilon \Delta$. Since there are $k_1^o k_3^o$ sources associated with each $b_5$-block (or $b_6$-block) and a total of $k_a^o$ sources, it follows that

$$\Pr\{A'\} = \frac{k_1^o k_3^o}{k_a^o}$$

Similarly, let B' be the event "the relation symbol r which a given $b_7$-block (or $b_6$-block) represents is associated with the target p", where $r \epsilon P$ and $p \epsilon \Pi$. Since there are $k_8^o k^o_{10}$ targets associated with each $b_7$-block (or $b_6$-block) and a total of $k_p^o$ targets, we obtain

$$\Pr\{B'\} = \frac{k_8^o k^o_{10}}{k_p^o}$$

Let A" be the event "one of the $m_{r_p}$ $b_6$-blocks (of the untransformed SSM) representing the relation symbol r and associated with the target p is associated with the source d", where $d \epsilon \Delta$, $r \epsilon P$, a $p \epsilon \Pi$.

Associated with the $b_1$-block representing the source d are $k_2^o k_4^o$ $b_6$-blocks representing distinct relation symbol. We wish to know the probability that one of these $b_6$-blocks coincides with one of the $m_{r_p}$ $b_6$-blocks representing the relation symbol r and associated with the $b_{11}$-block representing the target p. Since there are $m_6$ $b_6$-blocks which may be grouped into sets of $m_{r_p}$, we have $m_6/m_{r_p}$ sets from which to choose. Furthermore, since we have $k_2^o k_4^o$ opportunities for success,

$$\Pr\{A''\} = \frac{k_2^o k_4^o}{m_6/m_{r_p}}$$

$$= k_2^o k_4^o \, m_{r_p} \, \frac{k_1^o k_3^o}{k_2^o k_4^o k_a^o}$$

$$= m_{r_p} \, \frac{k_1^o k_3^o}{k_a^o}$$

Finally, let C be the event "the source d is associated with the target p", where $d \epsilon \Delta$ and $p \epsilon \Pi$.

Recall the-old-ball-in-the-urn trick: for $k = 0, 1, \cdots, n$, the probability of the event $E_k$ that one will score exactly k successes (where a success corresponds to drawing a red ball) when one draws without replacement a sample of size n from an urn containing M balls, of which m are red, is

$$\Pr\{E_k\} = \binom{n}{k} \frac{(m)_k (M-m)_{n-k}}{(M)_n}$$

where

$$\binom{n}{k} = \frac{(n)_k}{k!} = \frac{n!}{k!\,(n-k)!}$$

$$(M)_n = M(M-1)\cdots(M-n+1) \quad \text{and} \quad (M)_0 = 1$$

For our purposes here instead of containing balls, the urn will contain the sets of $m_{r_p}$ $b_6$-blocks which represent a given relation symbol associated with the $b_{11}$-block representing a particular target. The $k_7^o k_9^o / m_{r_p}$ sets of $b_6$-blocks which are associated with the $b_{11}$-block representing the target p will correspond to the red balls, and our sample will consist of the $k_2^o k_4^o$ $b_6$-blocks representing distinct relation symbols and associated with the $b_1$-block representing the source d. Then

$$M = \frac{k_2^o k_4^o k_a^o}{m_{r_p} k_1^o k_3^o}$$

$$m = \frac{k_7^o k_9^o}{m_{r_p}}$$

$$n = k_2^o k_4^o$$

$Pr\{C\}$ is then the probability that our sample of $k_2^o k_4^o$ $b_6$-blocks will contain a $b_6$-block from at least one of the $k_7^o k_9^o / m_{r_p}$ sets of $b_6$-blocks. Thus,

$$\begin{aligned} Pr\{C\} &= 1 - Pr\{E_o\} \\ &= 1 - \binom{n}{o} \frac{(m)_o (M-m)_n}{(M)_n} \\ &= 1 - \frac{(M-m)(M-m-1)\cdots(M-m-n+1)}{M(M-1)\cdots(M-n+1)} \end{aligned}$$

Clearly, the values assigned to m and n may be interchanged without affecting our result.

To digress for a moment, we may also apply this ball-in-the-urn concept to the event A''. In this case M is still given by

$$M = \frac{k_2^o k_4^o k_a^o}{m_{r_p} k_1^o k_3^o}$$

and n is still given by $n = k_2^o k_4^o$, but $m = 1$ since we are interested in only a single set of $m_{r_p}$ $b_6$-blocks. Furthermore, we wish to know the probability of exactly one success (which is the same as at least one success in this case, since there can be no more than one). Hence, $Pr\{A''\}$ will be given by

$$Pr\{A''\} = Pr\{E_1\}$$

$$= \binom{n}{1} \frac{(m)_1 (M-m)_{n-1}}{(M)_n}$$

$$= n \frac{m(M-m)(M-m-1)\cdots(M-m-n+2}{M(M-1)\cdots(M-n+1)}$$

$$= n \frac{(M-1)(M-2)\cdots(M-n+1)}{M(M-1)\cdots(M-n+1)}$$

$$= \frac{n}{M}$$

$$= k_2^o k_4^o \frac{m_{r_p} k_1^o k_3^o}{k_2^o k_4^o k_a^o}$$

$$= m_{r_p} \frac{k_1^o k_3^o}{k_a^o}$$

which is, of course, the result obtained before.

These results are summarized in Table 4-4, where for notational convenience we have introduced some new symbols $x_i$ for $i \epsilon \{1, 2, \cdots, 7\}$ to represent the various probabilities.*

* $x_6$ has been reserved for use in considering uniqueness of type 2, for which there must be defined an event B'' analogous to event A''.

$$x_1 = \Pr\{A\} = \frac{k_2^o k_4^o}{k_r^o}$$

$$x_2 = \Pr\{3\} = \frac{k_7^o \; k_9^o}{m_{r_p} \; k_r^o}$$

$$x_3 = \Pr\{A'\} = \frac{k_1^o k_3^o}{k_a^o}$$

$$x_4 = \Pr\{B'\} = \frac{k_8^o k_{10}^o}{k_p^o}$$

$$x_5 = \Pr\{A''\} = m_{r_p} \; \frac{k_1^o k_3^o}{k_a^o}$$

$$x_7 = \Pr\{C\} = 1 - \frac{(M-m)(M-m-1)\cdots(M-m-n+1)}{M(M-1)\cdots(M-n+1)}$$

$$\text{where } M = \frac{k_2^o k_4^o k_a^o}{m_{r_p} k_1^o k_3^o}$$

$$m = \frac{k_7^o k_9^o}{m_{r_p}}$$

$$n = k_2^o k_4^o$$

Table 4-4. Summary of Probabilities

Returning now to consideration of $t_{11}$, we see that the probabilities $y_1$, $y_2$, and $y_3$ are given by

$$y_1 = 1-x_1$$

$$y_2 = x_1(1-x_4)$$

$$y_3 = x_1 x_4$$

Let us rewrite the expression for $t_{11}$, incorporating these values for $y_1$, $y_2$, and $y_3$.

$$\begin{aligned} t_{11} = {} & T_a + (1-x_1)\, z_7\, (C_{r_1}) \\ & + x_1(1-x_4)\{(\bar{\Delta}_6+\bar{\Delta}_8)z^*_7(C_{r_1})+\Delta_6\Delta_8 z_7(C_{r_1})+m_{r_1}[\rho_1 z_3(C_p)+\bar{\rho}_1 z_4(C_p)]\} \\ & + x_1 x_4\{z^*_7(C_{r_1})+(\hat{m}_{r_1}-1)[\rho_1 z_3(C_p)+\bar{\rho}_1 z_4(C_p)] + [\rho_1 z^*_3(C_p)+\bar{\rho}_1 z^*_4(C_p)]\} \end{aligned}$$

Now let us consider the derivation of an expression for $t_{12}$, the number of time units required to perform $Q_1$ using its second method. This method is characterized by the following basic steps:

(1) Access all occurrences of the relation symbol $r_j$ in the storage structure.

(2) For each occurrence of the relation symbol $r_j$ examine all associated sources in search of $d_i$.

(3) If $d_i$ is found, examine all targets in the corresponding target set in search of $p_k$.

To initiate access of all occurrences of the relation symbol $r_j$, we must first access the head of the corresponding relation ring, which has a time cost of $T_r$ time units.

If $d_i$ is not associated with $r_j$, we will examine all sources associated with all occurrences of $r_j$ via $\alpha_5$ in search of $d_i$, which we will not find. Since the probability that $d_i$ is not associated with $r_j$ is $1-x_1$, the effective time cost for this situation will be $(1-x_1)z_5(C_a)$.

If $d_i$ is associated with $r_j$ but $p_k$ is not associated with this combination, we are faced with two possible situations. If there is a single copy of each $b_6$-block associated with the $b_1$-block(s) representing an arbitrary source (i.e., if $m_{r_1} = 1$), the $b_6$-block representing the relation symbol $r_j$ associated with the source $d_i$ may be determined by $\alpha_5^*$. If on the other hand $m_{r_1} > 1$, all $b_6$-blocks representing $r_j$ and all their associated $b_1$-blocks will have to be considered in the futile search for $p_k$. In other words $\alpha_5$ must be used to insure that no copy of a $b_6$-block representing $r_j$ has been overlooked. In both cases ($m_{r_1} = 1$ and $m_{r_1} > 1$), all $b_{11}$-blocks associated with each of the $m_{r_1}$ $b_6$-blocks must be examined via $\alpha_4$ in the search for $p_k$. Since the probability that $d_i$ is associated with $r_j$ but $p_k$ is not associated with that combination is $x_1(1-x_4)$, the effective time cost will be

$$x_1(1-x_4)[\,(\bar{\Delta}_6+\bar{\Delta}_8)z^*_5(C_a) + \Delta_6\Delta_8 z_5(C_a) + m_{r_1} z_4(C_p)]$$

Finally, if $d_i$ and $p_k$ are associated with the same $b_6$-block which represents $r_j$, we may use $\alpha^*_5$ to find the $b_1$-block and $b_6$-block representing the $d_i/r_j$ combination with which $p_k$ is associated. Since there are $m_{r_1}$ $b_6$-blocks which represent $r_j$ and which are associated with the $b_1$-block(s) representing $d_i$, we would expect $p_k$ to be associated with the $\hat{m}_{r_1}$-th $b_6$-block of these. Therefore, for $m_{r_1}$-1 of these blocks we will use $\alpha_4$ and for the $\hat{m}_{r_1}$-th block we will use $\alpha^*_4$. Our effective time cost for this situation will then be

$$x_1x_4[\,z^*_5(C_a) + (\hat{m}_{r_1}-1)z_4(C_p) + z^*_4(C_p)]$$

Combining all the terms obtained above yields the following expression for $t_{12}$:

$$\begin{aligned} t_{12} &= T_r + (1-x_1)z_5(C_a) \\ &+x_1(1-x_4)[\,(\bar{\Delta}_6+\bar{\Delta}_8)z^*_5(C_a)+\Delta_6\Delta_8 z_5(C_a)+m_{r_1} z_4(C_p)] \\ &+x_1x_4[\,z^*_5(C_a) + (\hat{m}_{r_1}-1)z_4(C_p) + z^*_4(C_p)] \end{aligned}$$

Next we will consider $t_{13}$, the number of time units required to perform $Q_1$ using its third method. Because of the symmetry of the SSM and the symmetry of methods 2 and 3 for $Q_1$, we would expect a very strong similarlity between the expressions for $t_{13}$ and $t_{12}$. In fact if $m_{r_p} = 1$, these expressions should be exact analogs. Let us then assume for the moment that $m_{r_p}$ is 1. The resulting expression for $t_{13}$ will be

$$
\begin{aligned}
t_{13} = T_r &+ (1-x_2)\, z_6\, (C_p) \\
&+x_2(1-x_5)[\, (\bar{\Delta}_3+\bar{\Delta}_5)z^*_6(C_p)+\Delta_3\Delta_5 z_6(C_p)+m_{r_2} z_2(C_a)] \\
&+x_2 x_5[\, z^*_6(C_p)+(\hat{m}_{r_2}-1)z_2(C_a)+z^*_2(C_a)\,]
\end{aligned}
$$

Let us now consider what effect $m_{r_p} \neq 1$ will have upon this expression. Quite simply if $m_{r_p} > 1$, there will be more than one $b_6$-block which represents the same realtion symbol associated with the $b_{11}$-block representing a given target. This means that when $p_k$ is associated with $r_j$ but $d_i$ is not associated with this combination, we must consider all $b_6$-blocks representing $r_j$ and all their associated $b_{11}$-blocks in the futile search for $d_i$ in order to insure that no $b_6$-block representing $r_j$ and associated with $p_k$ is overlooked. This is true regardless of whether or not $m_{r_2} = 1$. Secondly, instead of $m_{r_2}$ $b_6$-blocks to consider, we will have $m_{r_p} m_{r_2}$ $b_6$-blocks representing

$r_j$ to consider.

Let $\delta_p$ be a binary-valued variable, the value of which indicates whether $(\delta_p = 1)$ or not $(\delta_p = 0)$ $m_{r_p} = 1$. Also, let $m_{z_p} = m_{r_p} m_{r_2}$. Then we can rewrite the expression for $t_{13}$ to reflect the effects of $m_{r_p}$ as follows:

$$t_{13} = T_r + (1-x_2)z_6(C_p)$$

$$+x_2(1-x_5)[\delta_p(\bar{\Delta}_3+\bar{\Delta}_5)z^*_6(C_p)+(\bar{\delta}^*_p+\Delta_3\Delta_5)z_6(C_p)+m_{z_p}z_2(C_a)]$$

$$+x_2x_5[z^*_6(C_p)+(\hat{m}_{z_p}-1)z_2(C_a)+z^*_2(C_a)]$$

Finally, let us consider $t_{14}$, the number of time units required to perform $Q_1$ using its fourth method. Again, because of the symmetry of the SSM and the symmetry of methods 1 and 4 for $Q_1$, we would expect a strong similarity between the expressions for $t_{14}$ and $t_{11}$. The comments made about $m_{r_p}$ in our discussion of $t_{13}$ also apply here. Thus, we can easily derive the following expression for $t_{14}$:

$$t_{14} = T_p + (1-x_2)z_8(C_{r_2})$$

$$+x_2(1-x_5)\{\delta_p(\bar{\Delta}_3+\bar{\Delta}_5)z^*_8(C_{r_2})+(\bar{\delta}_p+\Delta_3\Delta_5)z_8(C_{r_2})+m_{z_p}[\rho_3 z_1(C_a)+\bar{\rho}_3 z_2(C_a)]\}$$

$$+x_2x_5\{z^*_8(C_{r_2})+(\hat{m}_{z_p}-1)[\rho_3 z_1(C_a)+\bar{\rho}_3 z_2(C_a)]+[\rho_3 z^*_1(C_a)+\bar{\rho}_3 z^*_2(C_a)]\}$$

As a matter of academic interest we should like to be assured that the probability that operation $Q_1$ is successful will be the same regardless of the method of implementation. Clearly, we are guaranteed that this is so if it is true that $x_1x_4$ equals $x_2x_5$. Let us check then whether the following equality is true:

$$x_1x_4 = x_2x_5$$

$$\frac{k^o_2k^o_4}{k^o_r} \quad \frac{k^o_8k^o_{10}}{k^o_p} = \frac{k^o_7k^o_9}{m_{r_p}\,k^o_r} \quad m_{r_p} \quad \frac{k^o_1k^o_3}{k^o_a}$$

$$k^o_2k^o_4k^o_8k^o_{10}k^o_a = k^o_1k^o_3k^o_7k^o_9k^o_p$$

which we know to be true.

We can continue in the manner above to develop expressions for the methods for each of the remaining primitive operations. Rather than belabor these derivations to the point of monotony, we have chosen simply to summarize the expressions for the various $t_{ij}$ in Appendix D.

## 4.3 Storage Cost Function

To complete our discussion of the measures of performance, let us develop an expression for the number of storage units required by a given storage structure.

If we can determine the number of storage units required for each type of block in a given structure, then we may sum over all block types the product of the number of storage units required for each type of block and the number of blocks of that type in the structure to determine the basic storage cost of the structure. For instance, if $u_i$ represents the number of storage units for each $b_i$-block, then the basic storage cost will be given by

$$\sum_{i=1}^{11} m_i u_i$$

To determine the entire storage cost of a structure we must add to this expression the number of storage units required by the various description blocks and the source, relation, and target ring heads. Let $u_d$ represent the (average) number of storage units required by a description block and let $u_a$, $u_r$, and $u_p$ represent the corresponding quantities for the source, relation, and target ring heads, respectively. The quantity to be added to the basic storage cost will then be given by

$$n\,u_d + k_a^o\,u_a + k_r^o\,u_r + k_p^o\,u_p$$

Thus, we may define the storage cost S for a given storage structure as

$$S = \sum_{i=1}^{11} m_i\,u_i + n\,u_d + k_a^o\,u_a + k_r^o\,u_r + k_p^o\,u_p$$

Let us consider now the determination of $u_i$ for all appropriate values of i. In general, a $b_i$-block consists of a number of fields containing pointers for the various rings which may pass through it, possibly a type field, and (if $i \epsilon \{5, 6, 7\}$) possibly a relation symbol name field. If we know the fields of which a $b_i$-block is composed, we can sum the numbers of storage units required by these fields to determine $u_i$.

Let us assume that a field containing a forward pointer for an $a_i$-ring requires $s_{f_i}$ units of storage and that a field containing a head pointer for an $a_i$-ring requires $s_{h_i}$ units of storage, where $i \epsilon \{1, 2, \cdots, 10\}$. Further, let us assume that a description block indicator requires $s_\delta$ units of storage and that a pointer in a source, relation, or target ring requires $s_p$ units. Finally, assume that a relation symbol name field requires $s_r$ units of storage and a block type field requires $s_t$ units. (If the type code does not require a separate field, we will assume that $s_t = 0$.)

As an example let us consider the derivation of an expression for $u_1$, the number of storage units required by a $o_1$-block. A block of this type contains a pointer field for the appropriate source ring; if $\phi_1 = 1$, it contains a pointer field for an $a_1$-ring; if $\phi'_1 = 1$, it contains a pointer field for a head pointer; if $\sigma_1 = 1$, it contains a pointer field for a description block indicator; and finally, if $\tau_1 = 1$, it contains a type field. As a result, $u_1$ may be characterized by the following expression:

$$u_1 = s_p + \phi_1 s_{f_1} + \phi'_1 s_{h_1} + \sigma_1 s_d + \tau_1 s_t$$

In a similar manner we may obtain expressions for all other block types.

$$u_{11} = s_p + \phi_{10} s_{f_{10}} + \phi'_{10} s_{h_{10}} + \sigma_2 s_d + \tau_{11} s_t$$

$$u_i = \beta_i(\phi_{i-1} s_{f_{i-1}} + \phi_i s_{f_i} + \tau_i s_t) \quad \text{for } i \epsilon \{2,4,8,10\}$$

$$u_i = \beta_i(\phi_{i-1} s_{f_{i-1}} + \phi'_{i-1} s_{h_{i-1}} + \phi_i s_{f_i} + \phi'_i s_{h_i} + \tau_i s_t)$$

$$\text{for } i \epsilon \{3,9\}$$

$$u_5 = \beta_5 (\phi_4 s_{f_4} + \phi'_4 s_{h_4} + \phi_5 s_{f_5} + \phi'_5 s_{h_5} + \rho_1 s_r + \tau_5 s_t)$$

$$u_6 = \beta_6 (\phi_5 s_{f_5} + \phi_6 s_{f_6} + s_p \rho_2 s_r + \tau_6 s_t)$$

$$u_7 = \beta_7 ( \phi_6 s_{f_6} + \phi'_6 s_{h_6} + \phi_7 s_{f_7} + \phi'_7 s_{h_7} + \rho_3 s_r + \tau_7 s_t )$$

We note that as a practical matter it is often the case that $s_{f_i}$, $s_{h_i}$, $s_d$, and $s_p$ are all equal (for all $i \epsilon \{1, 2, \cdots 10\}$).

# Chapter V

# A PROCEDURE FOR THE DETERMINATION OF A MINIMUM COST STORAGE STRUCTURE

In the previous chapter we developed two measures of performance - a time cost function T and a storage cost function S - for our Storage Structure Model. Our goal in this chapter will be to develop a procedure which utilizes these measures to determine a minimum cost storage structure for a given data structure. (Recall that we consider a storage structure to have minimum cost if it minimizes T subject to a given constraint on S.)

## 5.1 Reducing the Number of Feasible Solutions

T and S are functions of a large number of variables. In order to facilitate referring to these variables we will partition them into two classes: parametric variables (or simply, parameters) and decision variables. The parametric variables are those which characterize the environment in which our storage structure is to exist (e.g., $s_i$, $f_i$, $k_i^o$, and $s_{f_i}$) and are those over which we assume no control can be exercised to minimize T and S. The decision variables, on the other hand, characterize the form of the storage structure itself. In particular, the class of decision variables consists of $\phi_1 \cdots \phi_{10}$, $\phi'_1 \cdots \phi'_{10}$, $\Delta_1 \cdots \Delta_{10}$, $\beta_2 \cdots \beta_{10}$, $\sigma_1, \sigma_2, \rho_1, \rho_2$, and $\rho_3$. These are the variables over which the cost minimization is to be performed.

We know from our discussions of Chapter III that the values of the various decision variables may not all be specified independently. For instance, the values of $\beta_2 \cdots \beta_{10}$ are determined uniquely once values are assigned to $\Delta_1 \cdots \Delta_{10}$, $\rho_1, \rho_2$, and $\rho_3$. Let us assume for the moment, however, that the decision variables $\phi_1 \cdots \phi_{10}$, $\phi'_1 \cdots \phi'_{10}$, $\Delta_1 \cdots \Delta_{10}$, $\sigma_1, \sigma_2, \rho_1, \rho_2$, and $\rho_3$ are all independent. Since each of these variables is binary-valued, this implies that it is possible to specify $2^{35}$ or roughly $3 \times 10^{10}$ different storage structures via these variables.

Clearly, some caution must be exercised in choosing a method for determining that storage structure which satisfies our conditions of optimality lest solution of our problem become computationally infeasible.

Unfortunately, the situation is further complicated by the fact that the time cost function T is rather ill-behaved. First, T cannot feasibly be written as an explicit function of the decision variables. Second, T is a monotone function of none of the decision variables except $\rho_1, \rho_2$, and $\rho_3$. As a result, T does not lend itself to minimization by any of the common optimization techniques for functions of zero-one variables.

In order to reduce the number of "feasible solutions" which must be considered in determining the optimal storage structure, let us make a number of observations and decisions based upon these observations.

Consider for a moment an $a_i$-ring and $a_{i+1}$-ring pair where $i \epsilon \{2,4,6,8\}$. In general, this ring pair may assume any of the forms given in the following table:

| $a_i$-ring | $a_{i+1}$-ring | $\phi_i$ $\phi_{i+1}$ | $\Delta_i$ $\Delta_{i+1}$ |
|---|---|---|---|
| ring | ring | 1 1 | 0 0 |
| ring | stack | 1 0 | 0 0 |
| stack | ring | 0 1 | 0 0 |
| ring | duplicated | 1 0 | 0 1 |
| duplicated | ring | 0 1 | 1 0 |
| stack | duplicated | 0 0 | 0 1 |
| duplicated | stack | 0 0 | 1 0 |
| duplicated | duplicated | 0 0 | 1 1 |

We note that the time cost required to sequence through a stack is less than or equal to that required to sequence through a ring of the same composition. We also note that a ring and a stack have the same basic structure with regard to the ordering and accessability of their elements. Furthermore, a stack requires less storage than a ring of the same composition.

Since we are not concerned with manipulative operations (for which a ring can be preferable to a stack), it follows that the structures "ring-stack" and "stack-ring" for the $a_i$-ring and $a_{i+1}$-ring pair will always require less time for equivalent operations and less storage than the structure "ring-ring".

Since the remainder of the SSM is the same regardless of which of the three structures "ring-ring", "ring-stack", or "stack-ring" is implemented for the given ring pair, it is always advantageous to choose one of the latter two. We can therefore exclude $\phi_i=\phi_{i+1}=1$ for $i\epsilon\{2,4,6,8\}$ from further consideration.

Suppose that $\sigma_1=0$ and $\sigma_2=0$. Then each $a_1$-ring and $a_{10}$-ring pair functions just as the $a_i$-ring and $a_{i+1}$-ring pair just discussed (assuming that $\Delta$ and $\Pi$ are not disjoint). It follows that if $\sigma_1=0$ and $\sigma_2=0$, we can exclude $\phi_1=\phi_{10}=1$ from consideration. (Recall that when $\sigma_1=0$ and $\sigma_2=0$ at least one of $\phi_1$ and $\phi_{10}$ must be 1, however.)

If on the other hand $\sigma_1=1$ or $\sigma_2=1$ or both, then the $a_1$-rings and the $a_{10}$-rings will be independent of one another (as well as of the rest of the SSM). Applying the stack-versus-ring arguments to each in turn leads us to conclude that it is always advantageous to make the $a_1$-rings and the $a_{10}$-rings stacks instead of rings. Thus, when $\sigma_1=1$ or $\sigma_2=1$ or both, we should always set $\phi_1=0$ and $\phi_{10}=0$.

These observations have allowed us to reduce the number of combinations of values for $\phi_1\cdots\phi_{10}$ from 1024 to at most 81 when $\sigma_1=1$ or $\sigma_2=1$ or both and to at most 162 when $\sigma_1=0$ and $\sigma_2=0$.

Let us now examine the role of the head pointer in an arbitrary $a_i$-ring. If for a particular operation the direction of access is away from the head of the ring, the head pointer can make no contribution We can say in general then that if there are no operations to be performed upon the structure such that the direction of access is toward the head of the $a_i$-ring, there is no benefit to be obtained from a head pointer in that ring and we can set $\phi_i' = 0$. This will, of course, also tend to minimize the storage requirement of the structure.

Suppose on the other hand that there is at least one operation to be performed upon the structure such that the direction of access is toward the head of the $a_i$-ring. Since we expect to enter the ring at the $\hat{k}_i$-th element, either we can use the head pointer to go directly from this element to the head or we can step through the intervening $\hat{k}_i$-1 elements to reach the head, where the time required to step from one element to the next is either $s_i$ or $f_i$ depending upon whether the $a_i$-ring is a stack or an explicit ring, respectively. It follows that in order to accrue any benefit from a head pointer the time cost required to follow it must be less than or equal to that required to step through the intervening elements.

If the $a_i$-ring is an explicit ring ($\phi_i = 1$), this means that

$$h_i \leq \hat{k}_i f_i$$

must be true in order to gain advantage from $\phi_i' = 1$. (Recall our earlier assumption that $s_i \leq f_i \leq h_i$ for $i \in \{1, 2, \cdots 10\}$.)

If the $a_i$-ring is a stack ($\phi_i = 0$ and $\Delta_i = 0$), however, the situation becomes slightly more complex. Setting $\phi_i' = 1$ when $\phi_i = 0$ may mean that the time required to follow the head pointer is $h_i + s_o$ instead of $h_i$. (See the expressions for the various $z_i^o$.) Thus, if $\phi_i' = 1$ results in the inclusions of an $s_o$ term, the condition

$$h_i + s_o \leq \hat{k}_i s_i$$

must be satisfied for $\phi_i' = 1$ to result in a time reduction. On the other hand if no such inclusion results, the condition

$$h_i \leq \hat{k}_i s_i$$

must be met.

The result of this is that the values of $\phi_1' \cdots \phi_{10}'$ may be uniquely determined from the values of $\phi_1 \cdots \phi_{10}$ and $\Delta_1 \cdots \Delta_{10}$ (given actual values for $s_o$ and $s_i$, $f_i$, and $h_i$ for $i \epsilon \{1, 2, \cdots 10\}$), which reduces the number of solutions which must be considered by a factor of roughly $10^3$.

To further reduce the number of solutions which must be considered, let us treat the variables $\sigma_1, \sigma_2, \rho_1, \rho_2$, and $\rho_3$ as <u>external</u> decision variables. That is, let us assume that we assign values (externally to the solution process) to these variables and then solve for the optimal storage structure subject to the constraints of these values.

As a practical matter this assumption does not appear to be very restrictive since the overall optimal solution will probably result in the values

$$\begin{array}{cc} \underline{\sigma_1} & \underline{\sigma_2} \\ 0 & 1 \\ 1 & 0 \\ 0 & 0 \end{array} \quad \text{and} \quad \begin{array}{ccc} \underline{\rho_1} & \underline{\rho_2} & \underline{\rho_3} \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{array}$$

most of the time anyway. In any event all possible combinations can be considered if so desired by specifying them individually.

We are now faced with the problem of determining values for only $\phi_1 \cdots \phi_{10}$ and $\Delta_1 \cdots \Delta_{10}$ such that our optimality conditions are satisfied. It so happens that the constraints upon these variables (Chapter III) further reduce the number of solutions to be considered by a factor of 10.

In general terms we have reduced the number of solutions which must be considered by a factor of 10 via our decisions concerning $\phi_1 \cdots \phi_{10}$, a factor of $10^3$ by our decisions involving $\phi'_1 \cdots \phi'_{10}$, a factor of 30 for $\sigma_1, \sigma_2, \rho_1, \rho_2$, and $\rho_3$, and a factor of 10 from the constraints on $\phi_1 \cdots \phi_{10}$ and $\Delta_1 \cdots \Delta_{10}$. As a result we are faced now with considering only $10^4$ possible solutions. (In fact, when $\sigma_1$ and $\sigma_2$ are not both 0, we need consider at most 8960 solutions.)

Although we might wish to reduce this number further (which may in fact be impossible to accomplish through reasonable effort), it falls well within the computational limits of the computer and, hence, should not concern us unduly at this point.

We will, however, indicate a special case which allows us to reduce the number of solutions which must be considered by a factor of approximately 2. Because of the symmetry of the SSM, if we assign values to the various parameters and external decision variables in a symmetric manner (e.g., $k_1^o = k_{10}^o$, $k_2^o = k_9^o$, $\sigma_1 = \sigma_2$, $\rho_1 = \rho_3$, etc.), each solution will have a "mirror image" which has the same time and storage costs. For instance, if $\phi_1 \cdots \phi_{10} = 0001000010$ and $\Delta_1 \cdots \Delta_{10} = 0110111001$ is one solution and if the various parameters and external decision variables have symmetric values, then the mirror image solution $\phi_1 \cdots \phi_{10} = 0100001000$ and $\Delta_1 \cdots \Delta_{10} = 1001110110$ will have the same time and storage costs as the first solution.

This means that once we have considered a given solution we need not consider its mirror image. Note that certain solutions are their own mirror images. Hence, we must consider something over half of all the possible solutions.

## 5.2 Optimization Procedure

Let us now consider the basic procedure which we will use to determine the solution which satisfies our optimality conditions.

First we assign values to the various parameters and external decision variables. We then generate a sequence of all the value combinations for $\phi_1 \cdots \phi_{10}$. For each of these value combinations we generate a sequence of all those value combinations for $\Delta_1 \cdots \Delta_{10}$ which satisfy the constraints of Chapter III. For each resultant assignment of values to $\phi_1 \cdots \phi_{10}$ and $\Delta_1 \cdots \Delta_{10}$ we may determine the corresponding values of $\phi'_1 \cdots \phi'_{10}$ and $\beta_2 \cdots \beta_{10}$. A given combination of values for $\phi_1 \cdots \phi_{10}$, $\phi'_1 \cdots \phi'_{10}$, $\Delta_1 \cdots \Delta_{10}$, and $\beta_2 \cdots \beta_{10}$ describes a possible <u>state</u> of the SSM.

Our goal is to determine that state of the SSM which minimizes the time cost function T subject to a possible limit on the value of the storage cost function S.

Let $\eta_i$ represent the i-th state in the sequence of states, where $i \epsilon \{1, 2, \cdots, M\}$ and M is the number of states. Assume that the state $\eta_{i-1}$ is considered before the state $\eta_i$ which in turn is considered before the state $\eta_{i+1}$ and so forth.

Let $T_i$ and $S_i$ represent the values of T and S, respectively, for the state $\eta_i$.

Let $S_0$ represent the upper bound to be placed upon the storage cost function S.

We may then define the following procedure to determine the optimal state $\eta^*$ of the SSM.

(1) Set $T^* = \infty$ and $i = 1$.

(2) If $S_0 = \infty$, go to step (3).

Otherwise if $S_i > S_0$, go to step (4).

(3) If $T_i > T^*$, go to step (4).

If $T_i < T^*$, set $T^* = T_i$, $S^* = S_i$, $\eta^* = \eta_i$, and go to (4)

If $T_i = T^*$ and $S_i \leq S^*$, set $S^* = S_i$, $\eta^* = \eta_i$, and go to (4).

(4) Set $i = i+1$.

If $i \leq M$, go to step (2).

Otherwise, stop. $\eta^*$ is the optimal state of the SSM.

To illustrate the feasibility of this approach the procedure has been implemented via a prototype program on the Michigan Terminal System (MTS) for the IBM 360/67 at The University of Michigan. In the next chapter we will consider some results obtained through the use of this program.

# Chapter VI

# APPLICATION

In the preceding chapters we have developed a model for data structure, a model for the storage structures which can represent any data structure specified by the data structure model, and finally a technique for determining a minimum cost storage structure to represent a particular data structure in the solution of a given problem.

In this chapter we will apply these tools to a specific problem to determine a minimum cost storage structure for the data associated with the solution of that problem, we will compare the results we obtain to the storage structure used in an existing system for solving that problem, and we will examine the sensitivity of our results to variations in the parameters used to describe the problem and the structure of its data.

## 6.1 The Problem: A System for Medical Diagnosis

As the example to illustrate the use of the techniques we have developed, let us consider the representation of data associated with a computer system for aiding medical diagnosis.

In the next few sections which follow we will concern ourselves with a general description of a diagnostic system, the specification of the data and the data structure for a particular diagnostic system, and the determination of a minimum cost storage structure for that system.

### 6.1.1 General Description

In the general sense, the diagnostic problem is to ascertain the current state of some given system. The diagnostician uses information acquired from past experience with such systems, coupled with specific observations or tests of the given system, in order to deduce the identity of its current state.

In particular, the medical diagnostician is concerned with determining the "state" of his patient. The physician has learned through training and experience the sign and symptom patterns associated with possible diseases from which the patient can suffer.

In theory, a given set of signs and symptoms (hereafter referred to simply as symptoms) should characterize each disease uniquely.

In practice, however, different diseases may result in similar symptoms. Observation of many different symptoms may be required to identify a particular disease, and a given symptom may suggest many possible diseases. This fact, coupled with the large number of symptoms and diseases, requires the diagnostician to master considerable amounts of information.

The diagnostic process is further complicated by a number of other factors. First, the relationships between symptoms and diseases are often known only in probabilistic terms. Second, the tests required to determine whether indeed certain symptoms exist may exact a high cost (in terms of risk to the patient, patient discomfort, money, etc.). This cost must be weighed against the potential usefulness of the test results Finally, because of the vast amounts of information required in the diagnostic process, those symptom patterns and testing strategies associated with seldom encountered diseases may be effectively lost to (i.e., forgotten by) the diagnostician.

By using a computer program to provide general diagnostic assistance to its user, a number of these difficulties can be overcome, or at least minimized. One of the principal advantages to be gained from the use of a computer is the sheer bulk of information which it can maintain. Because a program can consider more possible diagnoses than a human being, it can provide a strong safeguard that a particular disease is not overlooked in the diagnosis.

For these reasons there has been increasing interest in computer-aided diagnosis. In particular, a number of programs have been written which are capable of performing diagnosis in particular medical areas. As a rule, these programs

employ a Bayesian analysis of symptoms based on a disease-symptom probability matrix for the given set of diseases considered. That is, the programs compute the probability of disease D given the symptom profile S (i.e., the set of symptoms) as follows:

$$P(D|S) = \frac{P(D)P(S|D)}{\sum_{D} P(D)P(S|D)}$$

where P(D) is the a priori probability of the occurrence of disease D and P(S|D) is the conditional probability that S occurs given D.

Since it is not our purpose to design a complete diagnostic system, we refer the reader to the literature for more complete descriptions of the mathematical techniques involved.

## 6.1.2 A Particular Diagnostic System

Gorry [25] has designed and implemented a general purpose diagnostic system (although the examples he has considered are exclusively in the area of medical diagnosis) which is representative of these systems and is the one which we shall use as a reference.

Let us examine the basic assumptions and characteristics of this system. The objective of the diagnostic process for the medical problem is to determine the malady or disease with which an individual is afflicted. This disease is assumed to be one of a finite, but perhaps quite large, number of possible diseases. Information concerning the given disease can be obtained by performing a variety of tests upon the patient. These tests may range from simple questions as in history-taking to complicated medical procedures such as exploratory surgery. The results of tests applied to the patient are then combined by the diagnostician with his experience with other diagnostic problems to deduce the particular disease. In the case of the diagnostic program this experience is reflected in probability distributions which characterize the results of certain tests given a particular disease. Specifically, this information relates symptoms (the results of the tests) to particular diseases. A symptom is assumed to be binary-valued (i.e., either present or absent) and a test is used to determine the presence or absence of some number (perhaps greater than one) of symptoms.

Associated with each test is a cost of applying it to the patient (in terms of risk or discomfort to the patient, the services of skilled personnel, money, etc.) and therefore it is advantageous to make a decision about the disease based upon a limited number of tests. On the other hand associated with each disease is a cost reflecting the loss resulting from diagnosing that disease as another. (Diagnosing a malignant tumor as benign is very costly ' should clearly be avoided.) The possibility of loss for an incorrect diagnosis tends to value extensive testing prior to making a decision. One must therefore balance the testing cost against the cost for an incorrect decision in an effort to minimize the overall expected cost.

To accomplish the goals above, Gorry's diagnostic system performs three logical functions:

(1) The interpretation of the symptoms for a particular problem via Bayes' rule, given conditional probabilities that particular symptoms are associated with certain diseases and given a priori probabilities that these diseases will occur in a given population. This is called the inference function.

(2) The selection of tests to be applied to the patient under consideration in order to obtain further symptoms. This is called the test selection function.

(3) The analysis of the symptoms observed to determine whether there are any irrelevant symptoms present (such as a sore elbow when all other symptoms indicate

typhoid fever) or, in the general case, to detect symptom patterns for more than one disease occuring simultaneously. This is called the pattern-sorting function.

The use of the inference function should be quite clear, but the test selection function and the pattern-sorting function may require brief descriptions.

Consider first the test selection function. The purpose of this function is to select tests in such a way as to minimize the overall expected loss of the diagnosis. For finite numbers of symptoms and diseases and a finite number of potentially useful test sequences the optimal choice of tests can be obtained by constructing a decision tree and folding back this tree in terms of expected loss. Such a tree - a portion of which appears in Figure 6-1 - consists of two types of nodes: decision nodes and what Gorry calls "nature's nodes".

A decision node represents the current status of the diagnostic problem as given by the probability distribution over the states (i.e., diseases) of the system. Emanating from each decision node are a number of branches, one for each possible test which can be performed and one corresponding to a terminal decision. Each branch corresponding to one of the tests which can be performed leads to one of nature's nodes.

Associated with each of nature's nodes are a number of branches

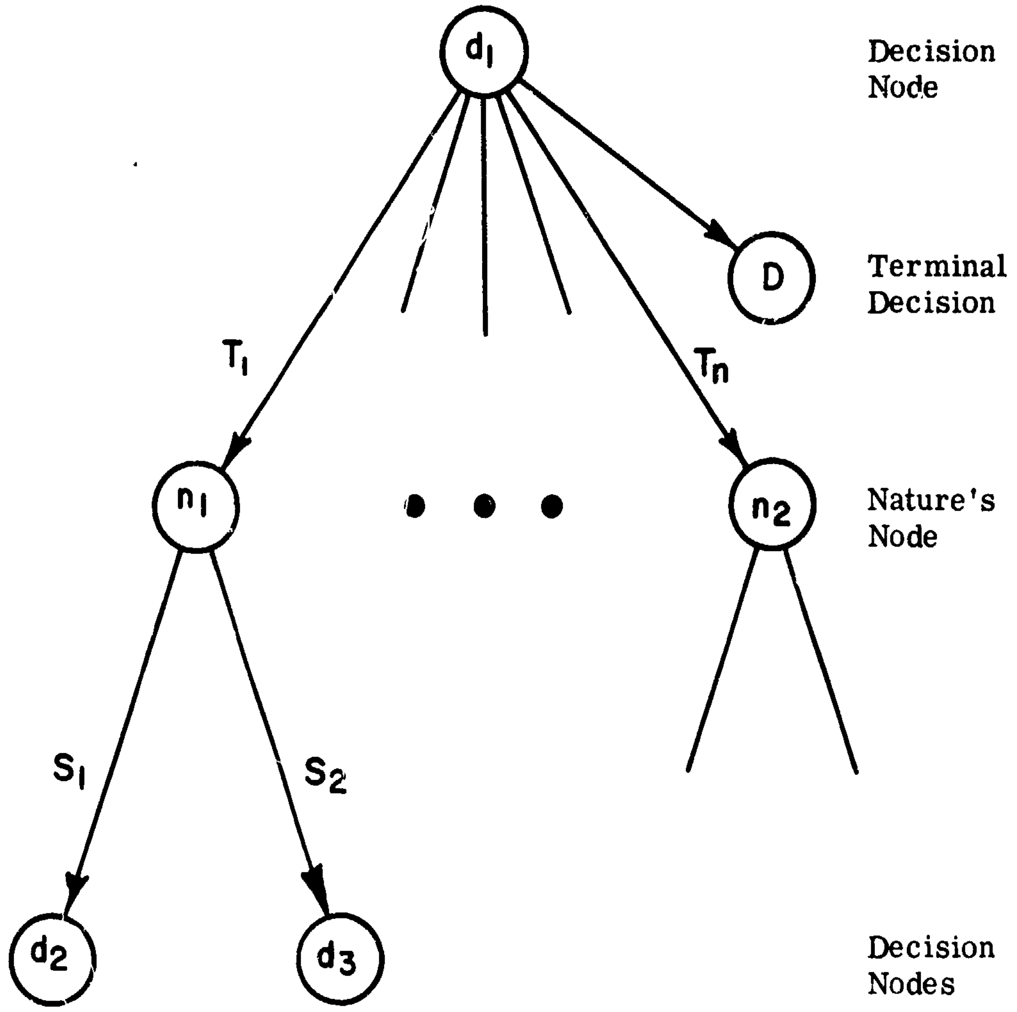


Figure 6-1. Section of a Decision Tree

representing the possible outcomes (i.e., symptoms) which can result from performing the test corresponding to the branch which leads to this node. Each of these "outcome branches" leads in turn to another decision node.

Thus, if somewhere in the course of performing a diagnosis we encounter decision node $d_1$ of Figure 6-1, we may choose to perform any one of the tests $T_1 \cdots T_n$ in order to gain further information about the state of the system under consideration (i.e., about the disease afflicting our patient), or we may choose to make a final diagnosis. If we decide to make a final diagnosis, we follow the branch of the decision tree which leads from decision node $d_1$ to the terminal decision D and the diagnostic process is concluded. Suppose on the other hand we decide to perform test $T_1$. In this case we follow the branch from decision node $d_1$ to nature's node $n_1$. Depending upon whether performing the test $T_1$ results in symptom $s_1$ or $s_2$, we will follow the branch from nature's node $n_1$ to decision node $d_2$ or decision node $d_3$, respectively. This newly accessed decision node, with the probability distribution of the states of the system updated to reflect the information conveyed by the symptom leading to the node, now represents the current states of the diagnosis. At this node we again have a choice between making a final diagnosis and performing one of several tests to obtain additional information.

Since the number of decision nodes of the decision tree grows exponentially with the number of symptoms and tests and since we may commonly expect to encounter problems involving large numbers of symptoms and tests, it is generally computationally infeasible to search the entire decision tree (or even a large portion of it) in order to determine the optimal sequence of tests to be performed to minimize the overall cost of the diagnosis . Therefore, Gorry has developed an heuristic which allows him to search a relatively small portion of the decision tree below each decision node encountered in the diagnostic process in order to determine which test should be performed at that node. While this heuristic results in a sub-optimal sequence of tests, the number of decision nodes which must be considered is greatly reduced, which more than offsets this disadvantage.

Starting with the root of the decision tree (i.e., the decision node which represents the initial status of the diagnostic process), the test selection function applies Gorry's heuristic to a portion of the decision tree below this node and determines the best test to perform at this stage of the diagnosis. The result obtained by performing this test leads to another decision node of the tree, at which point a terminal decision may be made or the test selection function may again be invoked. This process continues until a terminal decision (i.e., a final diagnosis) is reached.

Now consider the pattern-sorting function. As we shall see later only those symptoms significant to the diagnosis of a particular disease are associated with that disease by a probability greater than zero. For instance, the conditional probability of the symptom "sore elbow" given the disease "typhoid fever" is zero since a sore elbow is not a symptom of typhoid fever. If the symptom "sore elbow" were observed, in the course of a diagnosis in which typhoid fever was considered a possible cause of all the symptoms observed, the posterior probability for the disease typhoid fever (as computer by Bayes' rule) would be zero, and typhoid fever would be eliminated from further consideration even though all other symptoms strongly indicated the presence of this disease. The problem here is that while a sore elbow is not a symptom of typhoid fever, a patient certainly can have typhoid fever <u>and</u> a sore elbow. This is an example of the more general problem of irrelevant or "noise" symptoms. Unless special precautions are taken, such symptoms can eliminate the actual disease from consideration when processed by the inference function.

A second problem arises when symptoms associated with two or more distinct diseases are observed, as when the patient has more than one disease. In this case the diagnostic process must detect two or more patterns of symptoms, rather than dismissing certain symptoms as irrelevant.

Gorry's diagnostic system overcomes these problems by processing a number of symptom patterns in parallel during a diagnosis. A pattern is defined to consist of a subset of the set of symptoms observed to the current stage of the diagnosis, such that 1) at least one disease exhibits all the symptoms in the pattern with a non-zero probability, and 2) the pattern is not a subset of any other pattern. Each pattern, along with the probability distribution for the diseases of the patient given the symptoms of the pattern, is included in what Gorry calls a pattern stack*, which is maintained by the pattern-sorting function.

In addition to creating the pattern stack for the initial set of observed symptoms, the pattern-sorting function processes every new symptom against the pattern stack and updates the patterns accordingly. In particular, if a new symptom is relevant to at least one disease already indicated by a pattern, the symptom is added to the pattern and its probability distribution is updated. If no disease associated with a pattern exhibits the new symptom, no changes are made to either the pattern or its probability distribution. In addition, after the new symptom has been processed against all patterns in the pattern stack, the pattern-sorting function forms new patterns involving

* Gorry's use of the term "stack" in this case does not conform to our definition of the term. The pattern stack is actually implemented via a SLIP list.

the symptom if possible.

Finally, if the inference function determines that the probability of a particular pattern is zero, the pattern sorting function eliminates the pattern and its probability distribution from the pattern stack.

In general terms, Gorry's three functions interact in the following manner. The diagnostic system is provided with a list of symptoms which have been observed. The pattern-sorting function examines these, separates them into patterns, and sets aside for later consideration those patterns which do not appear to be relevant to the principal medical problem (i.e., the set of most likely diseases). Next, the inference function, using the a priori and conditional probabilities which constitute the "experience" of the diagnostic system, creates a probability distribution for the set of diseases which exhibit the symptoms of the pattern under consideration. Finally, the test selection function utilizes the current probabilities of the various diseases, the cost of each test, and the usefulness of the results of each test to select a good test to apply to the patient. (Alternatively, the test selection function may, of course, suggest a final diagnosis.) When the results of the test have been obtained, this process is repeated until a diagnosis has been obtained.

## 6.1.3 The Data and Its Data Structures

Let us now describe within the framework of our data structure model the basic data involved in the diagnostic process. The set of data items, $\Delta^o$, may be defined to consist of symptoms, diseases, and tests. We could also include the costs of the various tests and the a priori probabilities of the various diseases in the set of data items, but since there is a one-to-one relationship between costs and tests and between a priori probabilities and diseases, we choose to include the cost and the probabilities in each of the descriptions (i.e., definitions) of the tests and the diseases, respectively.

If we let $S^o = \{s_i | i=1,2,\cdots n_s\}$ represent the set of all symptoms, $D^o = \{d_i | i=1, 2, \ldots n_d\}$ the set of diseases, and $T^o = \{t_i | i=1, 2, \ldots n_t\}$ the set of possible tests, then $\Delta^o = S^o \cup D^o \cup T^o$.

The relations of interest may now be defined as follows:

$$r_1 = \{(d_i, t_j) | d_i \epsilon D^o, t_j \epsilon T^o, \text{ and test } t_j \text{ is relevant to disease } d_i\}$$

$$r_2 = \{(t_i, s_j) | t_i \epsilon T^o, s_j \epsilon S^o, \text{ and symptom } s_j \text{ is a possible result of test } t_i\}$$

$$r_j^o = \{(s_i, d_k) \mid s_i \in S^o,\ d_k \in D^o, \text{ and the conditional probability that symptom } s_i \text{ is present given disease } d_k \text{ is } x_j\}$$

where $x_j$ is an element of the set of conditional probabilities constituting the experience of the diagnostic system.

For these relations it is clear that $\Delta = \Pi = \Delta^o$. Note, however, that by reversing the order of the elements in the ordered pair $(s_i, d_k)$ for the relations $r_j^o$ we obtain $\Delta = D^o \cup T^o \subset \Delta^o$ and $\Pi = S^o \cup T^o \subset \Delta^o$. It may be advantageous to make this switch in order to reduce the cardinalities of $\Delta$ and $\Pi$, but for the purposes of our example we will assume the relations as are originally defined.

We have not included the costs of misdiagnosis in this discussion. The reason for excluding these costs is primarily that we choose to consider the determination of a representation for these items as a separate problem. To be more specific, we feel that the symptoms, diseases, and tests are very strongly interrelated (via relations $r_1$, $r_2$, and the various $r_j^0$) and, hence, should be considered together in determining a storage structure for their representation. On the other hand, we feel that the costs of misdiagnosis are not closely associated with any of these and, thus, should be considered alone in determining a storage structure for their representation. In other words, we feel that the costs of misdiagnosis have a data structure sufficiently distinct from the data structure for the symptoms, diseases, and tests to warrant separate consideration. This is then an example of a problem which we wish to partition into separate problems for the purposes of determining storage structures for the data involved.

Note that rather than actually determining a storage structure for the representation of the costs of misdiagnosis (via our procedure), we will assume simply that these data are represented in matrix form.

Let us now consider the actual diagnostic problem which we wish to use as our example. Warner and his associates [76] conducted a prolonged study of a number of types of congential heart disease, and as a result, they developed a disease-symptom probability matrix for

33 diseases (including "normal") and 50 symptoms*. The lists of symptoms and diseases appear in Tables 6-1 and 6-2, and the corresponding probability matrix appears in Table 6-3. Finally, the list of tests and their respective possible results appears in Table 6-4.

In an attempt to generalize this problem somewhat we will map all probabilities of the disease-symptom probability matrix which lie within the range $x - 0.025$ to $x + 0.025$ to the value x, where x is some multiple of 0.05. That is, to every probability y which satisfies the condition $x - 0.025 \leq y < x + 0.025$ we will assign a new value x. For example, 0.06 and 0.07 will become 0.05, and 0.08 and 0.09 will become 0.10.

Most of the probabilities in the matrix are already multiples of 0.05 so this has little effect upon the actual values therein (or their apparent validity). There are, however, a large number of entries with the values 0.01 and 0.02, and these will be mapped to 0. This should not adversely affect the diagnostic capability of the system and serves only to reduce the number of diseases which reflect a given symptom. It is hoped that the resultant interrelationships more accurately reflect those of a more general diagnostic problem, that is,

* In his consideration of this same problem Gorry apparently had access to Warner's updated matrix, for he considers 35 diseases and 57 attributes. The differences should be relatively slight, however.

| Symptom | Interpretation |
|---|---|
| S01 | Age, less than 1 year |
| S02 | Age, 1 year to 20 years |
| S03 | Age, 20 years or more |
| S04 | Cyanosis, mild |
| S05 | Cyanosis, severe (with clubbing) |
| S06 | Cyanosis, intermittent |
| S07 | Cyanosis, differential |
| S08 | Squatting |
| S09 | Dyspnea |
| S10 | Easy fatigue |
| S11 | Orthopnea |
| S12 | Chest pain |
| S13 | Repeated respiratory infections |
| S14 | Syncope |
| S15 | Systolic murmur, loudest at apex |
| S16 | Diastolic murmur, loudest at apex |
| S17 | Systolic murmur, loudest L 4th |
| S18 | Diastolic murmur, loudest L 4th |
| S19 | Continuous murmur, loudest L 4th |
| S20 | Systolic murmur with thrill, loudest L 2nd |
| S21 | Systolic murmur without thrill, loudest L 2nd |
| S22 | Diastolic murmur, loudest L 2nd |
| S23 | Continuous murmur, loudest L 2nd |
| S24 | Systolic murmur, loudest R 2nd |
| S25 | Diastolic murmur, loudest R 2nd |

Table 6-1. Symptoms for Congenital Heart Disease

| Symptom | Interpretation |
|---|---|
| S26 | Systolic murmur heard best posterior chest |
| S27 | Continuous murmur heard best posterior chest |
| S28 | Accentuated 2nd heart sound, L 2nd |
| S29 | Diminished 2nd heart sound, L 2nd |
| S30 | Right ventricular hyperactivity by palpation |
| S31 | Forceful apical thrust |
| S32 | Pulsatile liver |
| S33 | Absent or diminished femoral pulsation |
| S34 | ECG axis more than $110^o$ |
| S35 | ECG axis less than $0^o$ |
| S36 | R wave greater than 1.2 mv in lead $V_1$ |
| S37 | R' or qR pattern in lead $V_1$ |
| S38 | R wave greater than 2.0 mv in lead $V_6$ |
| S39 | T wave in lead $V_6$ inverted (no digitalis) |
| S40 | Early diastolic murmur, loudest at apex |
| S41 | Late diastolic murmur, loudest at apex |
| S42 | Holo-systolic murmur, loudest L 4th |
| S43 | Mid-systolic murmur, loudest L 4th |
| S44 | Holo-diastolic murmur, loudest L 4th |
| S45 | Early-diastolic murmur, loudest L 4th |
| S46 | Mid-systolic murmur with thrill, loudest L 2nd |
| S47 | Holo-systolic murmur without thrill, loudest L 2nd |
| S48 | Mid-systolic murmur without thrill, loudest L 2nd |
| S49 | Holo-systolic murmur without thrill, loudest L 2nd |
| S50 | Murmur louder than gr 3/6 |

Brackets indicate mutually exclusive symptoms.

Table 6-1. Symptoms for Congenital Heart Disease (Cont.)

| Disease | Interpretation |
|---|---|
| D01 | Normal |
| D02 | Atrial septal defect |
| D03 | Atrial septal defect with pulmonary stenosis |
| D04 | Atrial septal defect with pulomonary hyper-tension |
| D05 | Complete endocardial cushion defect |
| D06 | Partial anomalous pulmonary venous connections |
| D07 | Total anomalous pulmonary venous connections |
| D08 | Tricuspid atresia without transposition |
| D09 | Ebstein's anomaly |
| D10 | Ventricular septal defect with valvular pulmonary stenosis |
| D11 | Ventricular septal defect with infundibular stenosis |
| D12 | Pulmonary stenosis, valvular |
| D13 | Pulmonary stenosis, infundibular |
| D14 | Pulmonary atresia |
| D15 | Pulmonary artery stenosis |
| D16 | Pulmonary hypertension |
| D17 | Aortic-pulmonary window |
| D18 | Patent ductus arteriosus |
| D19 | Pulmonary arteriovenous fistula |
| D20 | Mitral stenosis |
| D21 | Primary myocardial disease |
| D22 | Anomalous origin of left coronary artery |
| D23 | Aortic valvular stenosis |
| D24 | Subaortic stenosis |
| D25 | Coarctation of aorta |

Table 6-2. Heart Disease Types

| Disease | Interpretation |
|---|---|
| D26 | Truneus arteriosus |
| D27 | Transposed great vessels |
| D28 | Corrected transposition |
| D29 | Absent aortic arch |
| D30 | Ventricular septal defect |
| D31 | Ventricular septal defect with pulmonary hypertension |
| D32 | Patent ductus arteriosus with pulmonary hypertension |
| D33 | Tricuspid atresia with transposition |

Table 6-2. Heart Disease Types (Cont.)

| | | SYMPTOMS | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| Diseases | Incidences | S01 | S02 | S03 | S04 | S05 | S06 | S07 | S08 |
| D01 | 0.100 | 01 | 49 | 50 | 01 | 00 | 01 | 00 | 01 |
| D02 | .081 | 10 | 50 | 50 | 02 | 01 | 02 | 00 | 01 |
| D03 | .005 | 30 | 60 | 10 | 20 | 10 | 20 | 00 | 01 |
| D04 | .001 | 10 | 20 | 70 | 30 | 10 | 25 | 00 | 01 |
| D05 | .027 | 20 | 50 | 30 | 15 | 05 | 10 | 00 | 01 |
| D06 | .005 | 10 | 40 | 50 | 01 | 01 | 01 | 00 | 01 |
| D07 | .001 | 20 | 70 | 10 | 65 | 10 | 05 | 00 | 01 |
| D08 | .018 | 50 | 48 | 02 | 30 | 65 | 01 | 00 | 10 |
| D09 | .001 | 10 | 45 | 45 | 22 | 44 | 01 | 00 | 22 |
| D10 | .054 | 40 | 55 | 05 | 25 | 25 | 10 | 00 | 30 |
| D11 | .063 | 40 | 55 | 05 | 30 | 30 | 10 | 00 | 40 |
| D12 | .045 | 20 | 70 | 10 | 01 | 01 | 01 | 00 | 01 |
| D13 | .013 | 20 | 70 | 10 | 01 | 01 | 00 | 00 | 01 |
| D14 | .014 | 90 | 09 | 01 | 10 | 90 | 00 | 00 | 80 |
| D15 | .001 | 05 | 45 | 50 | 01 | 01 | 01 | 00 | 01 |
| D16 | .013 | 10 | 45 | 45 | 01 | 01 | 01 | 00 | 01 |
| D17 | .001 | 30 | 60 | 10 | 05 | 01 | 01 | 00 | 01 |
| D18 | .072 | 20 | 40 | 40 | 01 | 01 | 01 | 00 | 01 |
| D19 | .002 | 20 | 30 | 50 | 45 | 45 | 01 | 00 | 01 |
| D20 | .008 | 20 | 50 | 30 | 01 | 01 | 01 | 00 | 01 |
| D21 | .013 | 70 | 29 | 01 | 01 | 01 | 01 | 00 | 01 |
| D22 | .001 | 70 | 29 | 01 | 01 | 01 | 01 | 00 | 01 |
| D23 | .036 | 10 | 80 | 10 | 01 | 01 | 01 | 00 | 01 |
| D24 | .009 | 10 | 80 | 10 | 01 | 01 | 01 | 00 | 01 |
| D25 | .054 | 10 | 70 | 20 | 01 | 01 | 01 | 00 | 01 |
| D26 | .005 | 50 | 40 | 10 | 30 | 60 | 01 | 00 | 15 |
| D27 | .063 | 90 | 10 | 00 | 20 | 60 | 05 | 10 | 05 |
| D28 | .001 | 30 | 30 | 30 | 30 | 05 | 10 | 00 | 01 |
| D29 | .001 | 60 | 39 | 01 | 01 | 01 | 01 | 80 | 30 |
| D30 | .252 | 15 | 70 | 15 | 01 | 01 | 01 | 00 | 01 |
| D31 | .081 | 30 | 60 | 10 | 30 | 50 | 10 | 00 | 05 |
| D32 | .005 | 30 | 40 | 30 | 01 | 01 | 05 | 50 | 01 |
| D33 | .009 | 40 | 55 | 05 | 50 | 20 | 10 | 00 | 01 |

Table 6-3. Symptom-Disease Probability Matrix

| Diseases | S09 | S10 | S11 | S12 | S13 | S14 | S15 | S16 |
|---|---|---|---|---|---|---|---|---|
| D01 | 01 | 10 | 03 | 05 | 05 | 03 | 05 | 01 |
| D02 | 35 | 50 | 05 | 02 | 40 | 01 | 02 | 02 |
| D03 | 60 | 70 | 05 | 02 | 10 | 10 | 02 | 02 |
| D04 | 80 | 90 | 05 | 05 | 15 | 10 | 02 | 02 |
| D05 | 40 | 50 | 05 | 05 | 30 | 05 | 60 | 15 |
| D06 | 15 | 20 | 01 | 05 | 05 | 01 | 02 | 02 |
| D07 | 70 | 80 | 05 | 05 | 20 | 05 | 02 | 02 |
| D08 | 80 | 90 | 20 | 05 | 15 | 10 | 02 | 05 |
| D09 | 80 | 80 | 10 | 30 | 15 | 22 | 05 | 25 |
| D10 | 75 | 90 | 05 | 05 | 10 | 20 | 02 | 02 |
| D11 | 75 | 90 | 05 | 05 | 10 | 25 | 02 | 02 |
| D12 | 50 | 65 | 01 | 01 | 01 | 10 | 02 | 02 |
| D13 | 50 | 65 | 01 | 01 | 01 | 10 | 02 | 02 |
| D14 | 90 | 99 | 05 | 10 | 05 | 35 | 02 | 02 |
| D15 | 01 | 01 | 01 | 01 | 01 | 01 | 04 | 01 |
| D16 | 70 | 95 | 40 | 10 | 10 | 10 | 01 | 01 |
| D17 | 10 | 10 | 05 | 01 | 10 | 01 | 05 | 10 |
| D18 | 20 | 20 | 10 | 01 | 10 | 05 | 05 | 15 |
| D19 | 10 | 20 | 05 | 01 | 01 | 10 | 05 | 02 |
| D20 | 50 | 50 | 40 | 05 | 10 | 10 | 80 | 20 |
| D21 | 40 | 50 | 20 | 01 | 05 | 05 | 15 | 02 |
| D22 | 30 | 30 | 30 | 80 | 15 | 20 | 05 | 01 |
| D23 | 20 | 30 | 20 | 15 | 01 | 35 | 20 | 02 |
| D24 | 20 | 30 | 20 | 15 | 01 | 35 | 20 | 02 |
| D25 | 20 | 30 | 20 | 01 | 01 | 05 | 05 | 01 |
| D26 | 15 | 30 | 05 | 01 | 20 | 10 | 02 | 02 |
| D27 | 60 | 70 | 20 | 01 | 05 | 10 | 05 | 02 |
| D28 | 10 | 20 | 01 | 01 | 01 | 01 | 05 | 02 |
| D29 | 10 | 50 | 05 | 20 | 01 | 20 | 05 | 02 |
| D30 | 20 | 30 | 05 | 01 | 15 | 05 | 05 | 20 |
| D31 | 60 | 70 | 20 | 10 | 20 | 10 | 05 | 01 |
| D32 | 20 | 30 | 10 | 01 | 10 | 05 | 02 | 02 |
| D33 | 80 | 90 | 20 | 01 | 30 | 05 | 05 | 10 |

Table 6-3. Symptom-Disease Probability Matrix (Cont.)

| Diseases | S17 | S18 | S19 | S20 | S21 | S22 |
|---|---|---|---|---|---|---|
| D01 | 70 | 02 | 07 | 00 | 80 | 01 |
| D02 | 30 | 20 | 02 | 05 | 90 | 02 |
| D03 | 05 | 05 | 02 | 57 | 40 | 01 |
| D04 | 15 | 20 | 02 | 05 | 40 | 20 |
| D05 | 90 | 40 | 02 | 10 | 20 | 10 |
| D06 | 20 | 02 | 02 | 02 | 60 | 05 |
| D07 | 10 | 15 | 10 | 05 | 75 | 05 |
| D08 | 65 | 05 | 05 | 20 | 20 | 02 |
| D09 | 95 | 25 | 05 | 05 | 15 | 02 |
| D10 | 20 | 02 | 05 | 65 | 25 | 02 |
| D11 | 20 | 02 | 05 | 65 | 25 | 02 |
| D12 | 10 | 02 | 05 | 70 | 20 | 02 |
| D13 | 10 | 02 | 02 | 70 | 20 | 02 |
| D14 | 40 | 05 | 05 | 01 | 02 | 02 |
| D15 | 02 | 01 | 01 | 02 | 25 | 02 |
| D16 | 30 | 05 | 01 | 01 | 05 | 30 |
| D17 | 20 | 05 | 60 | 01 | 10 | 05 |
| D18 | 10 | 02 | 50 | 02 | 13 | 05 |
| D19 | 10 | 02 | 20 | 02 | 10 | 02 |
| D20 | 10 | 10 | 02 | 05 | 10 | 02 |
| D21 | 05 | 02 | 02 | 02 | 05 | 02 |
| D22 | 01 | 01 | 01 | 01 | 01 | 01 |
| D23 | 20 | 10 | 02 | 05 | 05 | 01 |
| D24 | 20 | 10 | 02 | 05 | 05 | 01 |
| D25 | 20 | 10 | 02 | 02 | 10 | 01 |
| D26 | 70 | 02 | 02 | 10 | 10 | 02 |
| D27 | 50 | 02 | 02 | 03 | 10 | 02 |
| D28 | 70 | 02 | 02 | 05 | 30 | 02 |
| D29 | 50 | 02 | 02 | 10 | 30 | 02 |
| D30 | 95 | 05 | 02 | 10 | 10 | 05 |
| D31 | 50 | 10 | 02 | 05 | C5 | 25 |
| D32 | 10 | 10 | 02 | 02 | 20 | 10 |
| D33 | 70 | 05 | 02 | 10 | 30 | 10 |

Table 6-3. Symptom-Disease Probability Matrix (Cont.)

| Diseases | S23 | S24 | S25 | S26 | S27 | S28 | S29 | S30 |
|---|---|---|---|---|---|---|---|---|
| D01 | 05 | 01 | 00 | 01 | 01 | 15 | 05 | 10 |
| D02 | 02 | 01 | 01 | 01 | 01 | 60 | 01 | 80 |
| D03 | 03 | 01 | 01 | 01 | 02 | 30 | 15 | 40 |
| D04 | 01 | 01 | 01 | 01 | 01 | 95 | 01 | 50 |
| D05 | 01 | 01 | 01 | 01 | 01 | 70 | 02 | 40 |
| D06 | 05 | 01 | 01 | 10 | 15 | 40 | 02 | 10 |
| D07 | 20 | 01 | 01 | 10 | 15 | 85 | 02 | 80 |
| D08 | 05 | 01 | 01 | 01 | 01 | 02 | 60 | 01 |
| D09 | 05 | 01 | 01 | 01 | 01 | 02 | 35 | 10 |
| D10 | 05 | 02 | 02 | 10 | 15 | 10 | 60 | 20 |
| D11 | 05 | 02 | 02 | 10 | 15 | 10 | 60 | 20 |
| D12 | 10 | 02 | 02 | 01 | 01 | 10 | 60 | 20 |
| D13 | 02 | 02 | 02 | 01 | 01 | 10 | 60 | 20 |
| D14 | 05 | 02 | 02 | 10 | 10 | 01 | 90 | 20 |
| D15 | 01 | 20 | 02 | 50 | 05 | 10 | 02 | 10 |
| D16 | 02 | 02 | 02 | 02 | 02 | 95 | 00 | 30 |
| D17 | 20 | 02 | 02 | 02 | 02 | 70 | 01 | 20 |
| D18 | 85 | 02 | 02 | 03 | 05 | 50 | 01 | 20 |
| D19 | 05 | 01 | 01 | 05 | 70 | 05 | 05 | 20 |
| D20 | 02 | 02 | 02 | 01 | 01 | 50 | 01 | 20 |
| D21 | 02 | 10 | 02 | 01 | 01 | 20 | 02 | 10 |
| D22 | 01 | 01 | 01 | 01 | 01 | 20 | 02 | 01 |
| D23 | 01 | 95 | 05 | 01 | 01 | 20 | 10 | 01 |
| D24 | 01 | 95 | 05 | 01 | 01 | 20 | 10 | 01 |
| D25 | 05 | 15 | 10 | 80 | 15 | 10 | 10 | 01 |
| D26 | 02 | 02 | 02 | 05 | 10 | 40 | 10 | 30 |
| D27 | 02 | 05 | 02 | 01 | 01 | 20 | 10 | 20 |
| D28 | 02 | 05 | 02 | 01 | 01 | 20 | 10 | 10 |
| D29 | 02 | 05 | 02 | 01 | 01 | 90 | 02 | 40 |
| D30 | 01 | 02 | 05 | 01 | 01 | 30 | 02 | 05 |
| D31 | 01 | 02 | 05 | 01 | 01 | 90 | 02 | 30 |
| D32 | 02 | 02 | 02 | 02 | 02 | 90 | 02 | 30 |
| D33 | 02 | 02 | 01 | 01 | 01 | 30 | 10 | 01 |

Table 6-3. Symptom-Disease Probability Matrix (Cont.)

| Diseases | S31 | S32 | S33 | S34 | S35 | S36 | S37 |
|---|---|---|---|---|---|---|---|
| D01 | 03 | 01 | 01 | 01 | 02 | 02 | 02 |
| D02 | 01 | 01 | 01 | 70 | 05 | 05 | 85 |
| D03 | 01 | 05 | 01 | 85 | 05 | 20 | 70 |
| D04 | 01 | 05 | 01 | 85 | 05 | 20 | 70 |
| D05 | 10 | 10 | 01 | 05 | 70 | 05 | 85 |
| D06 | 01 | 01 | 01 | 15 | 02 | 02 | 15 |
| D07 | 01 | 01 | 01 | 90 | 02 | 25 | 75 |
| D08 | 20 | 30 | 01 | 02 | 90 | 02 | 02 |
| D09 | 20 | 10 | 01 | 10 | 02 | 02 | 60 |
| D10 | 01 | 02 | 01 | 95 | 02 | 85 | 10 |
| D11 | 01 | 02 | 01 | 95 | 02 | 85 | 10 |
| D12 | 01 | 05 | 01 | 95 | 02 | 85 | 10 |
| D13 | 01 | 05 | 01 | 95 | 02 | 85 | 10 |
| D14 | 01 | 02 | 01 | 95 | 02 | 85 | 10 |
| D15 | 01 | 01 | 01 | 10 | 02 | 10 | 02 |
| D16 | 01 | 10 | 01 | 95 | 02 | 90 | 05 |
| D17 | 40 | 01 | 01 | 01 | 15 | 02 | 02 |
| D18 | 40 | 02 | 01 | 02 | 10 | 02 | 02 |
| D19 | 01 | 01 | 01 | 05 | 05 | 02 | 02 |
| D20 | 05 | 02 | 01 | 50 | 02 | 10 | 40 |
| D21 | 50 | 02 | 01 | 05 | 10 | 05 | 05 |
| D22 | 05 | 01 | 01 | 05 | 10 | 05 | 05 |
| D23 | 40 | 01 | 05 | 05 | 15 | 02 | 02 |
| D24 | 40 | 01 | 05 | 05 | 15 | 02 | 02 |
| D25 | 30 | 01 | 99 | 05 | 05 | 02 | 02 |
| D26 | 05 | 01 | 01 | 30 | 10 | 40 | 10 |
| D27 | 20 | 02 | 02 | 40 | 20 | 30 | 05 |
| D28 | 10 | 01 | 01 | 20 | 10 | 10 | 10 |
| D29 | 05 | 01 | 10 | 70 | 05 | 80 | 05 |
| D30 | 30 | 01 | 01 | 30 | 10 | 05 | 05 |
| D31 | 05 | 05 | 01 | 70 | 05 | 75 | 15 |
| D32 | 05 | 05 | 01 | 70 | 05 | 75 | 15 |
| D33 | 20 | 30 | 01 | 02 | 90 | 02 | 02 |

Table 6-3. Symptom-Disease Probability Matrix (Cont.)

| Diseases | S38 | S39 | S40 | S41 | S42 | S43 | S44 | S45 |
|---|---|---|---|---|---|---|---|---|
| D01 | 02 | 02 | 01 | 00 | 02 | 70 | 04 | 03 |
| D02 | 02 | 02 | 01 | 02 | 01 | 30 | 02 | 20 |
| D03 | 02 | 02 | 01 | 01 | 01 | 05 | 01 | 05 |
| D04 | 02 | 02 | 01 | 02 | 01 | 15 | 20 | 02 |
| D05 | 02 | 02 | 15 | 01 | 85 | 05 | 02 | 20 |
| D06 | 02 | 02 | 02 | 02 | 02 | 20 | 02 | 02 |
| D07 | 02 | 02 | 02 | 02 | 30 | 10 | 01 | 30 |
| D08 | 90 | 10 | 05 | 02 | 50 | 15 | 05 | 02 |
| D09 | 02 | 02 | 25 | 25 | 45 | 45 | 25 | 25 |
| D10 | 02 | 02 | 02 | 02 | 20 | 05 | 02 | 02 |
| D11 | 02 | 02 | 02 | 02 | 20 | 05 | 02 | 02 |
| D12 | 02 | 02 | 01 | 01 | 01 | 10 | 02 | 02 |
| D13 | 02 | 02 | 01 | 01 | 01 | 10 | 01 | 01 |
| D14 | 02 | 02 | 02 | 01 | 30 | 40 | 02 | 05 |
| D15 | 02 | 02 | 01 | 01 | 02 | 02 | 01 | 00 |
| D16 | 02 | 02 | 01 | 01 | 01 | 30 | 15 | 05 |
| D17 | 60 | 05 | 10 | 02 | 10 | 20 | 05 | 02 |
| D18 | 50 | 05 | 10 | 02 | 05 | 10 | 02 | 02 |
| D19 | 02 | 02 | 02 | 02 | 10 | 10 | 02 | 02 |
| D20 | 02 | 02 | 20 | 20 | 10 | 10 | 10 | 10 |
| D21 | 40 | 90 | 02 | 02 | 10 | 10 | 02 | 02 |
| D22 | 20 | 90 | 01 | 01 | 01 | 01 | 01 | 01 |
| D23 | 70 | 15 | 02 | 02 | 02 | 20 | 10 | 02 |
| D24 | 70 | 15 | 02 | 02 | 02 | 20 | 10 | 02 |
| D25 | 40 | 04 | 01 | 01 | 05 | 20 | 10 | 02 |
| D26 | 20 | 05 | 02 | 02 | 40 | 40 | 02 | 02 |
| D27 | 20 | 05 | 02 | 02 | 30 | 30 | 02 | 02 |
| D28 | 10 | 10 | 02 | 02 | 30 | 30 | 02 | 02 |
| D29 | 10 | 05 | 02 | 02 | 30 | 30 | 02 | 02 |
| D30 | 15 | 05 | 20 | 02 | 92 | 05 | 05 | 01 |
| D31 | 10 | 05 | 01 | 01 | 30 | 30 | 10 | 02 |
| D32 | 10 | 05 | 02 | 02 | 10 | 10 | 02 | 02 |
| D33 | 90 | 10 | 10 | 02 | 30 | 30 | 05 | 05 |

Table 6-3. Symptom-Disease Probability Matrix (Cont.)

| Diseases | S46 | S47 | S48 | S49 | S50 |
|---|---|---|---|---|---|
| D01 | 00 | 00 | 80 | 05 | 10 |
| D02 | 05 | 01 | 90 | 01 | 60 |
| D03 | 60 | 01 | 38 | 01 | 70 |
| D04 | 05 | 01 | 40 | 01 | 40 |
| D05 | 02 | 20 | 20 | 20 | 80 |
| D06 | 02 | 02 | 60 | 02 | 30 |
| D07 | 05 | 01 | 80 | 02 | 70 |
| D08 | 20 | 20 | 20 | 20 | 50 |
| D09 | 15 | 15 | 05 | 05 | 50 |
| D10 | 60 | 05 | 25 | 05 | 90 |
| D11 | 60 | 05 | 25 | 05 | 90 |
| D12 | 68 | 01 | 25 | 01 | 80 |
| D13 | 68 | 01 | 25 | 01 | 80 |
| D14 | 01 | 01 | 02 | 02 | 20 |
| D15 | 02 | 01 | 25 | 02 | 60 |
| D16 | 02 | 02 | 05 | 02 | 20 |
| D17 | 02 | 02 | 10 | 05 | 75 |
| D18 | 05 | 02 | 20 | 10 | 85 |
| D19 | 02 | 02 | 10 | 10 | 30 |
| D20 | 05 | 05 | 10 | 10 | 70 |
| D21 | 02 | 02 | 05 | 05 | 10 |
| D22 | 01 | 01 | 01 | 01 | 10 |
| D23 | 05 | 01 | 05 | 01 | 90 |
| D24 | 05 | 01 | 05 | 01 | 90 |
| D25 | 02 | 02 | 10 | 05 | 65 |
| D26 | 10 | 10 | 10 | 10 | 40 |
| D27 | 03 | 03 | 10 | 10 | 50 |
| D28 | 05 | 05 | 30 | 30 | 60 |
| D29 | 10 | 10 | 30 | 30 | 20 |
| D30 | 01 | 10 | 01 | 10 | 85 |
| D31 | 01 | 05 | 01 | 05 | 50 |
| D32 | 02 | 02 | 20 | 20 | 20 |
| D33 | 10 | 10 | 30 | 30 | 50 |

Table 6-3. Symptom-Disease Probability Matrix (Cont.)

| Test | Possible Results |
|---|---|
| T01 | S01, S02, S03 |
| T02 | S04, S05, S06, S07, N |
| T03 | S08, N |
| T04 | S09, N |
| T05 | S10, N |
| T06 | S11, N |
| T07 | S12, N |
| T08 | S13, N |
| T09 | S14, N |
| T10 | S15, N |
| T11 | S16, N |
| T12 | S17, S18, S19, N |
| T13 | S20, S21, S22, S23, N |
| T14 | S24, N |
| T15 | S25, N |
| T16 | S26, S27, N |
| T17 | S28, S29, N |
| T18 | S30, N |
| T19 | S31, N |
| T20 | S32, N |
| T21 | S33, N |
| T22 | S34, S35, N |
| T23 | S36, S37, N |
| T24 | S38, N |
| T25 | S39, N |
| T26 | S40, S41, N |
| T27 | S42, S43, N |
| T28 | S44, S45, N |
| T29 | S46, S47, S48, S49, N |
| T30 | S50, N |

N means "normal"

Table 6-4. Tests for Heart Disease Diagnosis

one which encompasses a greater variety of diseases and symptoms.

Using this new probability matrix along with the numbers of symptoms, diseases, and tests, we will determine values for the parameters which characterize our data structure model: namely, $k^o_a$, $k^o_p$, $k^o_r$, $m_{r_p}$, and $k^o_1 \cdots k^o_{10}$.

Since $|S^o| = 50$, $|D^o| = 33$, and $|T^o| = 30$, we know that $k^o_a = k^o_p = 113$. From the probability matrix we see that there are 20 relations $r^o_j$: $r^o_1$ for 0.05, $r^o_2$ for 0.10, $\cdots$, and $r^o_{20}$ for 1.00. Therefore, $k^o_r = 22$.

For each "type" of relation (i.e., $r^o_j$, $r_1$, and $r_2$) we have compiled certain statistics which appear in Table 6-5. If the totals for items 1 and 2 and those for items 3 and 4 are divided by the number of sources and the number of targets, respectively, and if the number of sources and the number of targets are divided by the totals for items 5 and 6, respectively, we obtain the averages contained in Table 6-6.

We can also determine from inspection of our given information that there are no identical (shared) target sets ($\Pi^2_\ell$ 's) and no two sources have a relation/target set pair in common. These two facts imply that $k^o_7 = 1$ and $k^o_3 = 1$, respectively. The fact that $k^o_3 = 1$ implies that $k^o_2 = 1$, also.

Items 5 and 6 of Table 6-6 correspond to $k^o_1$ and $k^o_{10}$, respectively.

| | | $s_i r^0_j d_k$ | $d_i r_1 t_j$ | $t_i r_2 s_j$ | Total |
|---|---|---|---|---|---|
| 1) | $\sum_{\text{sources}}$ {number of relation/target pairs associated with a given source} | 937 | 636 | 50 | 1623 |
| 2) | $\sum_{\text{sources}}$ {number of distinct relations associated with a given source} | 332 | 33 | 30 | 395 |
| 3) | $\sum_{\text{targets}}$ {number of source/relation pairs associated with a given target} | 937 | 636 | 50 | 1623 |
| 4) | $\sum_{\text{targets}}$ {number of distinct relations associated with a given target} | 327 | 30 | 50 | 407 |
| 5) | number of source subsets ($\Delta^1_\ell$'s) | 50 | 33 | 30 | 113 |
| 6) | number of target subsets ($\Pi^1_m$'s) | 33 | 24 | 30 | 87 |

Table 6-5. Statistics for Heart Disease Problem

| | | |
|---|---|---|
| 1) | Average number of relation/ target pairs associated with a given source | 14.36 |
| 2) | Average number of distinct relations associated with a given source | 3.50 |
| 3) | Average number of source/ relation pairs associated with a given target | 14.36 |
| 4) | Average number of distinct relations associated with a given target | 3.60 |
| 5) | Average number of sources in a given source subset | 1.00 |
| 6) | Average number of targets in a given target subset | 1.30 |

Table 6-6. Averages for Heart Disease Problem

Therefore, $k_1^o = 1$ and $k_{10}^o = 1.$[illegible]$0$. Item 2 of Table 6-6 corresponds to the product $k_2^o k_4^o$ and since $k_2^o = 1$, we find that $k_4^o = 3.50$. Item 1 represents the product $k_2^o k_4^o k_6^o k_8^o k_{10}^o$, and since we know the values of all these factors except $k_8^o$, we can determine that $k_8^o = 3.13$. Similarly, item 3 represents the product $k_1^o k_3^o k_5^o k_7^o k_9^o$ from which we determine that $k_9^o = 14.36$.

This leaves us with only $m_{r_p}$ remaining to be determined. Recall that $m_{r_p}$ represents the expected number of times the same relation symbol is associated with a given target. Since $k_7^o k_9^o = 14.36$, we know that associated with each target there are on the average 14.36 relations which are not necessarily distinct. Item 4 of Table 6-6 indicates, however, that there are an average of only 3.60 distinct relations associated with a given target. Hence, $m_{r_p} = 14.36/3.60 = 3.99$.

For convenience all of these results are summarized in Table 6-7.

Since our prototype program uses fixed-point arithmetic in most of the low-level routines, we require that the information of Table 6-7 be in integer form. This implies transformation of these results to integer values approximating the original values and satisfying the given constraints. (Clearly, we may not simply round off the values.) Table 6-8 contains one possible choice for the transformed values.

| | |
|---|---|
| $k_a^o =$ | 113 |
| $k_p^o =$ | 113 |
| $k_r^o =$ | 22 |
| $m_{r_p} =$ | 3.99 |
| $k_1^o =$ | 1 |
| $k_2^o =$ | 1 |
| $k_3^o =$ | 1 |
| $k_4^o =$ | 3.50 |
| $k_7^o =$ | 1 |
| $k_8^o =$ | 3.13 |
| $k_9^o =$ | 14.36 |
| $k_{10}^o =$ | 1.30 |

Table 6-7. Summary of Parameter Values for Heart Disease Problem

| | |
|---|---|
| $k_a^o =$ | 113 |
| $k_p^o =$ | 113 |
| $k_r^o =$ | 22 |
| $m_{r_p} =$ | 4 |
| $k_1^o =$ | 1 |
| $k_2^o =$ | 1 |
| $k_3^o =$ | 1 |
| $k_4^o =$ | 3 |
| $k_7^o =$ | 1 |
| $k_8^o =$ | 3 |
| $k_9^o =$ | 18 |
| $k_{10}^o =$ | 2 |

Table 6-8. Transformed Parameters for Heart Disease Problem

## 6.1.4 The Operations

We are now concerned with estimation of the relative frequencies of the operations to be used in the diagnostic process.

In pattern sorting two basic operations appear: namely, (1) determine the diseases associated with a given symptom, and (2) determine the symptoms associated with a given disease. The first of these is used to determine those diseases which may be indicated by the set of symptoms in a given pattern, and the second is then used to determine the intersection of the symptoms associated with each of these diseases with the set of given symptoms. (Note that if each time the first operation is performed the symptom leading to a particular disease is recorded for that disease, then there is no need for the second operation.)

Gorry indicates that his program, operating in the sequential diagnosis mode, was given on the average 7 initial symptoms and performed an average of 5.8 tests (each of which yields one symptom as a result). We may, therefore, assume that the first operation above is performed on the average 12.8 times in the course of a solution.

Since each symptom can indicate on the average 18.7 diseases (from the symptom-disease matrix), we can probably assume that most, if not all, diseases must be considered for the second operation. We will assume conservatively that 25 diseases are involved and, hence, that the second operation is performed 25 times in the course of a solution.

These two operations correspond to our primitive operations $Q_{12}(d_i{*}\text{-})$ and $Q_{14}(\text{-}{*}p_k)$, respectively. Thus, $Q_{12}$ is performed on the average 12.8 times and $Q_{14}$, 25 times.

The inference process (i.e., the application of Bayes' rule) also involves two basic operations: (1) determine the a priori probability of a given disease, and (2) determine the conditional probability that a particular symptom is associated with a given disease. Since we have assumed that the a priori probability of a given disease is part of the disease description itself (and, hence, is contained within the disease description block), we need not concern ourselves with the first operation as it cannot affect the structure.

The inference function processes each observed symptom against every disease in the pattern stack. If the observed symptom is relevant to at least one disease associated with a given pattern, then Bayes' rule is applied to each disease associated with the pattern. The second operation must therefore be applied once for every such disease and the given observed symptom.

Since we have no way of validly estimating the average number of patterns in the pattern stack (although Gorry does assure us that this number should ordinarily be quite limited) but we do know that the process should eventually result in only one pattern (unless the patient has two or more unrelated diseases, which we assume he does not), we will assume that the pattern stack always contains only one pattern. Our next problem is to determine the number of diseases associated with the pattern on the average. Since each symptom can indicate an average of 18.7 diseases, we will assume that the initial pattern consisting of the first observed symptoms has that many diseases associated with it. At the end of the diagnostic process, Gorry's statistics for this problem show an average of 4.1 diseases having probabilities greater than a given threshold (0.01). Let us assume then that the pattern remaining at the end of the diagnosis has 4.1 diseases associated with it. At the risk of offending the purist let us assume that the average number of diseases associated with a pattern throughout the course of the diagnosis is the average of the average numbers for the initial and final patterns, which is 11.4.

Since on the average 12.8 symptoms are considered in the course of the diagnosis, we can say that the second operation, which corresponds to primitive operation $\phi_7(d_i\text{-}p_k)$, is performed (12.8)(11.4) = 145.9 times.

The test selection function determines the set of tests which are relevant to the diseases whose posterior probabilities at the current decision node each exceed a given threshold (i.e., the most likely diseases so far). Excluded are those tests which may already have been considered. Then for each possible result of each test the inference function is invoked. The resulting probabilities along with the costs of tests and the loss function matrix are used to determine which test should actually be performed.

We see then that test selection involves the following basic operations: (1) determine the tests relevant to the given diseases, (2) determine the possible outcomes of each test, (3) determine the cost of each test, (4) determine certain elements of the loss matrix, and the inference function basic operations.

Since we have assumed that the cost of a given test is part of the test description, we need not concern ourselves with its determination. Since the loss function matrix is not part of the structure under consideration (it is a matrix), we may ignore determination of its elements also. Thus, we need concern ourselves only with determining the tests and their outcomes and the conditional probabilities that particular symptoms are associated with given disease.

Gorry's results indicate that for this problem an average of 350 decision nodes are encountered in the solution process. Let us assume as we did in considering the inference process that there are an average of 11.4 diseases of interest at each decision node. Let us further assume that only 3 of these diseases have posterior probabilities greater than the given threshold. (It would be unlikely that all the diseases are equally probable.) From Tables 6-3 and 6-4 we can determine that there are an average of 19.3 tests which are pertinent to each disease and each test has an average of 1.7 possible outcomes (symptoms) other than "normal".

Then the first of the test selection operations will be performed 3 times at each decision node, or a total of 1050 times.

Although each disease has 19.3 tests associated with it on the average, there may be considerable overlap between the sets of tests for two different diseases. Let us assume that the 3 diseases of interest at each decision node have 22 distinct tests associated with them out of the 30 possible tests. Since some of these may already have been considered (by the end of the solution process a total of 12.8 will have been considered), let us assume that 15 remain to be considered at any given node. Then the second of the test selection operations will be performed 15 times for each decision node, or a total of 5250 times.

Note that these two operations are both of the form of primitive operation $Q_3(d_i r_j -)$. Therefore, $Q_3$ will be performed a total of 6300 times during the solution process.

Since each test has an average of 1.7 outcomes, there will be on the average $(1.7)(15) = 25$ symptoms to be considered at each decision node, which implies that operation $Q_7$ (of the inference function) must be performed 75 times at each node, or a total of 26,250 times. $Q_7$ will then be performed a grand total of 26,396 times.

To summarize, the following primitive operations are involved in the solution process with the indicated (rounded-off) absolute and relative frequencies:

| Operation | Frequency | Relative Frequency |
|---|---|---|
| $Q_3$ $(d_i r_j -)$ | 6300 | 0.192 |
| $Q_7$ $(d_i - p_k)$ | 26396 | 0.806 |
| $Q_{12}(d_i * -)$ | 13 | 0.001 |
| $Q_{14}(- * p_k)$ | 25 | 0.001 |
| | 32734 | |

## 6.1.5 The Environment

As a final step before submitting our problem to the solution process and determining the optimal storage structure for it, we must assign values to the unspecified parameters and the external decision variables.

As for the external decision variables, let us assume $\sigma_1 = 0$, $\sigma_2 = 1$, $\rho_1 = 0$, $\rho_2 = 1$, and $\rho_3 = 0$. The choice of these values is largely an arbitrary one, but it does have an intuitive justification. Since the sets $\Delta$ and $\Pi$ are not disjoint, we must handle the "wrap-around" problem by either requiring at least one of $\sigma_1$ and $\sigma_2$ to be non-zero or requiring some $\phi_i$ to be non-zero. Rather than constrain the $\phi_i$'s in this manner, we choose to set $\sigma_2 = 1$. Since for the problem under consideration there appears to be no advantage to be gained from setting both $\sigma_1$ and $\sigma_2$ to one, we choose to set $\sigma_1 = 0$.

The values assigned to $\rho_1$, $\rho_2$, and $\rho_3$ are simply the "natural" choices. A relation symbol name appears only once for each use when $\rho_1 = 0$, $\rho_2 = 1$ and $\rho_3 = 0$, and it is equally accessible from both sources and targets.

Consider now the various unspecified parameters. First let us assume that our structure is uniform in nature. That is, let us assume that all pointers in the structure occupy the same size fields and that they all require the same number of time units to follow; let us assume that all stepping operations for stacks require the same number of time units for execution; and so forth. More specifically, let the following conditions be true:

$$f_1 = \cdots = f_{10} = h_1 = \cdots = h_{10} = f_a = f_p = F_a = F_r = F_p$$

$$s_o = s_1 = \cdots = s_{10}$$

$$c_d = c_r \text{ and } v_d = v_r$$

$$T_a = T_r = T_p$$

$$u_a = u_p$$

$$s_p = s_d = s_{f_1} = \cdots = s_{f_{10}} = s_{h_1} = \cdots = s_{h_{10}}$$

For further convenience let us also assume that $T_a$, $T_r$, and $T_p$ represent simply the times required to follow pointers and, hence, are equal to $F_a$, $F_r$, $F_p$, etc.

Not included in the conditions above, but also in need of values are the parameters $u_d, u_r, s_r, s_t$, and $S_o$.

In order to rigorously specify the environment (i.e., the computer) in which the diagnostic system is to reside and operate, we will assume it to be an IBM 360/67, with which we assume the reader to be familiar. With this in mind let us briefly examine the characteristics of each of the quantities to which we must assign a value.

The act of following a pointer simply involves replacing the value of the position indicator (described in Chapter IV) by the contents of the field to which that value points, where this field is assumed, of course, to contain a pointer. By letting the position indicator occupy one of the general registers of the 360/67, this operation can be implemeted via the load instruction (L $R_1, D_2(X_2, B_2)$), which has an average execution time of 1.20 $\mu$s (microseconds) [34].

Stepping from one field to another in a stack involves incrementing or decrementing the value of the position indicator by the size of the blocks in the stack. Assuming that one of the general registers contains this size, the operation can be implemented via an RR type add or subtract instruction (AR R1, R2 or SR R1, R2, respectively), each of which has an average execution time of 0.65 μs.

The comparison operations characterized by $c_d$ and $c_r$ involve comparing the contents of the field to which the position indicator points with some given quantity and selecting the next operation to be performed on the basis of the outcome of this comparison. If we assume the given quantity is contained a general register, we can use a compare instruction (C R1,D2(X2,B2)) followed by a branch-on-condition instruction (BC M1,D2(X2,B2)) to implement the operation. The compare has an average execution time of 1.40 μs and the branch on condition has an average execution time between 0.80 μs and 1.10 μs, depending upon the frequency with which the branch is successful.

The "fetch" operations characterized by $v_d$ and $v_r$ are used to move the contents of the field to which the position indicator points to some other memory location. If one of the general registers contains the address of the location to which the given quantity is to be transferred, we can implement the operation by using a load instruction (L R1,D2(X2,B2)) followed by a store instruction (ST R1,D2(X2,B2)), which have average execution times of 1.20 μs and 0.93 μs, respecively.

Clearly, there is a certain amount of overhead associated with each of these operations which we have not considered - such as, the amount of time involved in program "looping" when an operation or a sequence of operations is to be performed recursively. Our analysis should, however, give us a good estimate of the relative times required to perform each of these operations.

If we normalize and round-off the time quantities indicated above, we find that a pointer operation requires 4 time units, a stack operation requires 2 time units, a compare requires 8 time units, and a fetch requires 7 time units. Thus,

$$f_1 = \cdots = f_{10} = h_1 = \cdots = h_{10} = f_a = f_p = F_a = F_r = F_p = T_a = T_r = T_p = 4$$

$$s_o = s_1 = \cdots = s_{10} = 2$$

$$c_d = c_r - 8$$

$$v_d = v_r = 7$$

Consider now those parameters which pertain to storage. Since an address on the IBM 360 is 24 bits in length, we could assume a pointer field to be 3 bytes in length. For program efficiency, however, we should require the right-hand end of this field to be aligned on a full word boundary. This is best accomplished by using a 4-byte (or full word) field. Therefore, we will assume pointer fields to be 4 bytes in length. The left most (unused) byte can be used for type information. Thus,

$$s_p = s_d = s_{f_1} = \cdots = s_{f_{10}} = s_{h_1} = \cdots = s_{h_{10}} = 4$$

$$s_t = 0$$

By assuming that the source and target ring heads consist simply of a pointer, we also have

$$u_a = u_p = 4$$

Since 20 of the 22 relations of interest in our diagnostic problem are probabilities, we will assume that the relation symbol name fields of our structure are large enough to contain the actual values of the probabilities to which these relations correspond. In particular, this must be 4 bytes. Therefore,

$$s_r = 4$$

We will assume that the relation ring heads consist of a relation symbol name field and pointer. Therefore,

$$u_r = 8$$

The description block corresponding to a symptom need contain only the name or a code representing that symptom. For this, a description block consisting of 4 bytes is probably sufficient. On the other hand, since each test has a cost associated with it and each disease has a probability, the description blocks corresponding to them should be on the order of 3 bytes. Let us assume, therefore, that the average description block consists of 6 bytes. Thus,

$$u_d = 6$$

Finally, since the IBM 360/67 is a virtual memory machine, we can assume that there is no limit (within reason, or course) to the amount of memory available for our structure. This means that

$$S_o = \infty$$

We now have a complete description of the environment within which our optimal storage structure must reside.

### 6.1.6 The Solution

We are now ready to proceed with determination of the optimal storage structure.

In our search for the optimal solution we partition (rather arbitrarily) the set of feasible solutions into a number of families of solutions, one for each distinct combination of values assigned to $\phi_1 \cdots \phi_{10}$. Each of these so-called ϕ-families contain all the feasible solutions which have a particular combination of values for $\phi_1 \cdots \phi_{10}$.

In addition to determining the overall optimal solution (or solutions), we also determine the best solution (or solutions) within each of the ϕ-families and this information is recorded along with the overall optimum. Appendix E contains a listing of these results for the problem currently under consideration, which we will designate Case 1.

Upon inspection of this information we find that there are six equally good solutions for our problem, four of them in $\phi$-family number 10 and two in $\phi$-family number 28. [ The number assigned to a $\phi$-family simply indicates its position in the sequence of $\phi$-families as they are generated and is used to facilitate referring to individual ones. ] Table 6-9 contains more complete descriptions of these solutions.

Let us examine each of the solutions in turn, starting with Solution 1 of $\phi$-family 28. Given the values for the decision variables and the values for $k_1 \cdots k_{10}$ we can generate the schematic representation of Figure 6-2.

In this schematic each type of block which has not been eliminated from the structure is represented by a single field regardless of the actual number of fields which that block may contain. The double-walled field at the top of the structure represents a description block. The solid arrows emanating from the $b_3$-blocks and the $b_4$-blocks correspond to the pointers of the $a_3$-rings, and the dashed arrows originating from the $b_3$-blocks represent head pointers. The arrows originating from the $b_7$-blocks correspond to $a_6$-ring head pointers and, of course, the arrows from the $b_{11}$-blocks represent description block indicators. The source, relation, and target rings - although present - are not shown.

$\phi$- family: 10

Solution 1

$\phi_1 \cdots \phi_{10}$: 0 0 0 0 1 0 0 0 0 0

$\phi'_1 \cdots \phi'_{10}$: 0 0 0 0 1 1 0 0 0 0

$\Delta_1 \cdots \Delta_{10}$: 1 0 1 1 0 0 1 1 1 1

$\beta_2 \cdots \beta_{10}$: 0 0 1 0 0 0 1 1 1

$k_1 \cdots k_{10}$: 1 3 1 1 1 6 1 1 1 1

Solution 2

$\phi_1 \cdots \phi_{10}$: 0 0 0 0 1 0 0 0 0 0

$\phi'_1 \cdots \phi'_{10}$: 0 0 0 0 1 0 0 1 0 0

$\Delta_1 \cdots \Delta_{10}$: 1 0 1 1 0 1 1 0 1 1

$\beta_2 \cdots \beta_{10}$: 0 0 1 0 0 1 0 0 1

$k_1 \cdots k_{10}$: 1 3 1 1 1 1 1 6 1 1

Table 6-9. Case 1 Optimal Solutions

**Solution 3**

| | |
|---|---|
| $\phi_1 \cdots \phi_{10}$: | 0 0 0 0 1 0 0 0 0 0 |
| $\phi'_1 \cdots \phi'_{10}$: | 0 0 0 0 1 1 0 0 0 0 |
| $\Delta_1 \cdots \Delta_{10}$: | 1 1 1 0 0 0 1 1 1 1 |
| $\beta_2 \cdots \beta_{10}$: | 1 1 0 0 0 0 1 1 1 |
| $k_1 \cdots k_{10}$: | 1 1 1 3 1 6 1 1 1 1 |

**Solution 4**

| | |
|---|---|
| $\phi_1 \cdots \phi_{10}$: | 0 0 0 0 1 0 0 0 0 0 |
| $\phi'_1 \cdots \phi'_{10}$: | 0 0 0 0 1 0 0 1 0 0 |
| $\Delta_1 \cdots \Delta_{10}$: | 1 1 1 0 0 1 1 0 1 1 |
| $\beta_2 \cdots \beta_{10}$: | 1 1 0 0 0 1 0 0 1 |
| $k_1 \cdots k_{10}$: | 1 1 1 3 1 1 1 6 1 1 |

Table 6-9. Case 1 Optimal Solutions (Cont.)

$\phi$-family: 28

Solution 1

| | |
|---|---|
| $\phi_1 \cdots \phi_{10}$: | 0 0 1 0 0 0 0 0 0 0 |
| $\phi'_1 \cdots \phi'_{10}$: | 0 0 1 0 0 1 0 0 0 0 |
| $\Delta_1 \cdots \Delta_{10}$: | 1 0 0 1 1 0 1 1 1 1 |
| $\beta_2 \cdots \beta_{10}$: | 0 0 0 1 0 0 1 1 1 |
| $k_1 \cdots k_{10}$: | 1 3 1 1 1 6 1 1 1 1 |

Solution 2

| | |
|---|---|
| $\phi_1 \cdots \phi_{10}$: | 0 0 1 0 0 0 0 0 0 0 |
| $\phi'_1 \cdots \phi'_{10}$: | 0 0 1 0 0 0 0 1 0 0 |
| $\Delta_1 \cdots \Delta_{10}$: | 1 0 0 1 1 1 1 0 1 1 |
| $\beta_2 \cdots \beta_{10}$: | 0 0 0 1 0 1 0 0 1 |
| $k_1 \cdots k_{10}$: | 1 3 1 1 1 1 1 6 1 1 |

Table 6-9. Case 1 Optimal Solutions (Cont.)

Items Common to All Six Solutions

$m_a = 1$ $m_p = 18$ $m_{r_1} = 1$ $m_{r_2} = 1$

| Operation | Method | Cost |
|---|---|---|
| $Q_3$: $d_i r_j -$ | $1(\hat{d}\ \hat{r}\ p)$ | $t_{3,1} = 116.00$ |
| $Q_7$: $d_i - p_k$ | $1(\hat{d}\ r\ \hat{p})$ | $t_{7,1} = 285.22$ |
| $Q_{12}$: $d_i * -$ | $1(\hat{d}\ p)$ | $t_{12,1} = 272.00$ |
| $Q_{14}$: $- * p_k$ | $1(\hat{p}\ d)$ | $t_{14,1} = 530.00$ |

$T = 252.96$ $S = 33398$

Table 6-9. Case 1 Optimal Solutions (Cont.)

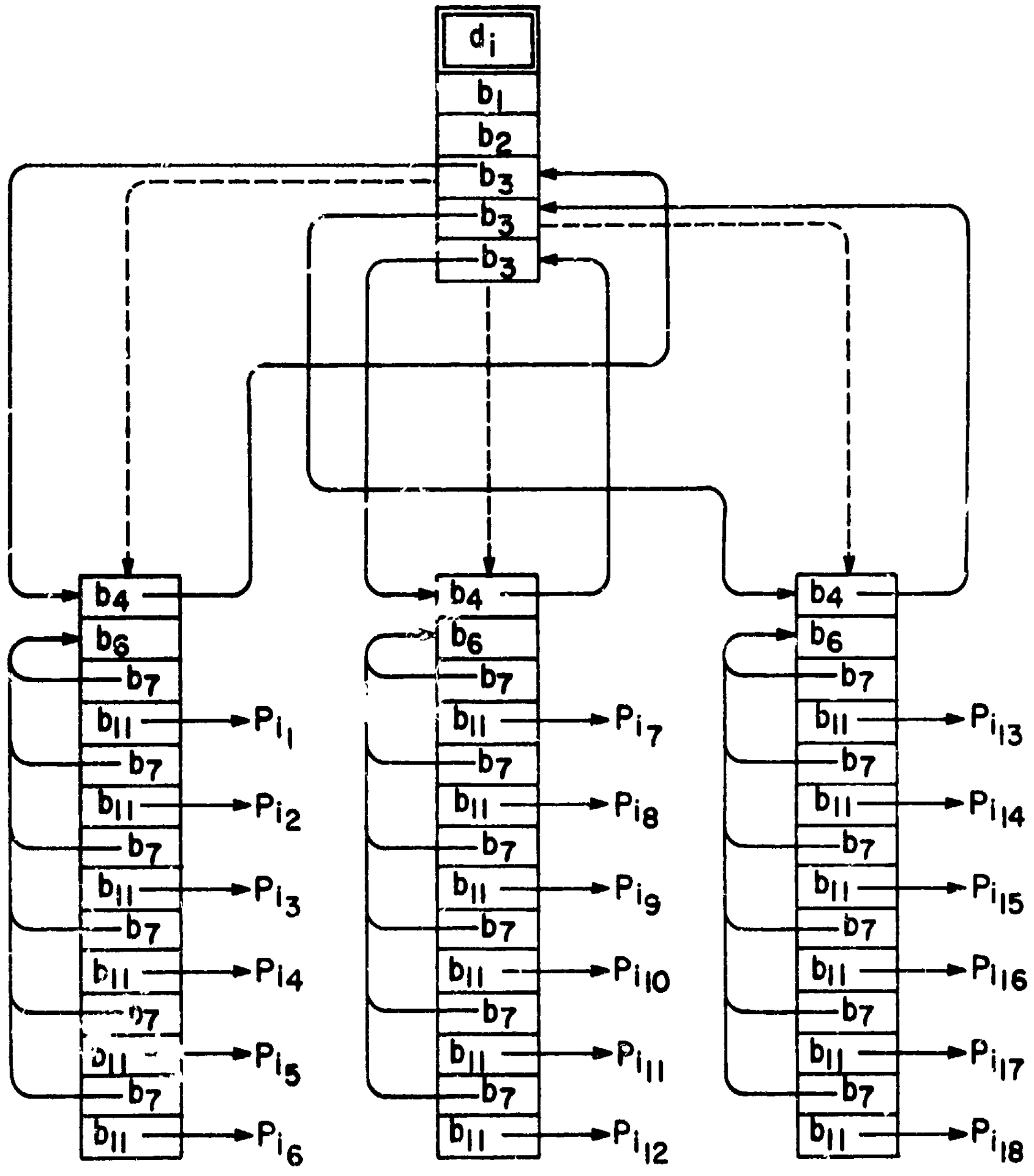


Figure 6-2. Schematic of Solution 1 for Case 1

Since $k_3 = 1$, the $a_3$-ring head pointer might appear to be super-fluous - the forward pointer from the $b_3$-block to the $b_4$-block [illegible]ars to perform the same function - but in general this is not the case. We earlier assumed that a head pointer points to the next item of [illegible]rect interest as we sequence through the structure. In this case the next item of interest when following the head pointer or moving in that direction is the $b_6$-block (and its relation symbol name field). On the other hand the $a_3$-ring forward pointer simply points to the next forward pointer of the ring; in order to access the corresponding $b_6$-block we must step from this pointer field to the $b_6$-block at a cost of $s_0$ time units. It should be clear then that the $a_3$-ring head pointer performs a function distinct from the forward pointer which parallels it.

If we assume, however, that the environment in which the structure is implemented is a computer utilizing the base-displacement form of addressing (ala the IBM 360), then the forward pointer can perform the same function as the head pointer (in addition to its original function), in which case the head pointer may be omitted. (This would not be true, of course, if $k_3$ were greater than 1.)

If desired, we could include in our procedure the facility for eliminating from those rings of the storage structure for which $k_i = 1$ any head pointers which might otherwise arise. We do not ch[illegible] to do so here and, hence, will maintain the structure as given.

Replacing each block by the fields it contains yields the actual storage structure which appears in Figure 6-3 sans source, relation, and target ring pointers.

Since the methods used to implement operations $Q_3$, $Q_7$, and $Q_{12}$ use the source ring as starting points and the method used to implement operation $Q_{14}$ uses the target rings (none of the operations uses the relation rings) and since the $b_6$-blocks contain a relation symbol name field (hence, the relation rings need never be used to determine the relation represented by a given block), the relation rings are not really needed in the structure. We may assume, therefore, that they are not present.

Furthermore, since $m_a = 1$, it is not necessary for the source rings to be closed loops. That is, a $b_1$-block need not contain a pointer back to the head of the corresponding source ring.

Again, we could include in our procedure the steps necessary to automatically account for these contingencies. We did not feel that the effort required to do so for this and other such cases was warranted for a prototype program, however.

Thus, the only addition required to make Figure 6-3 more accurately reflect the actual storage structure is the inclusion of a target ring pointer for every description block indicator. (We leave this to the imagination of the reader.)

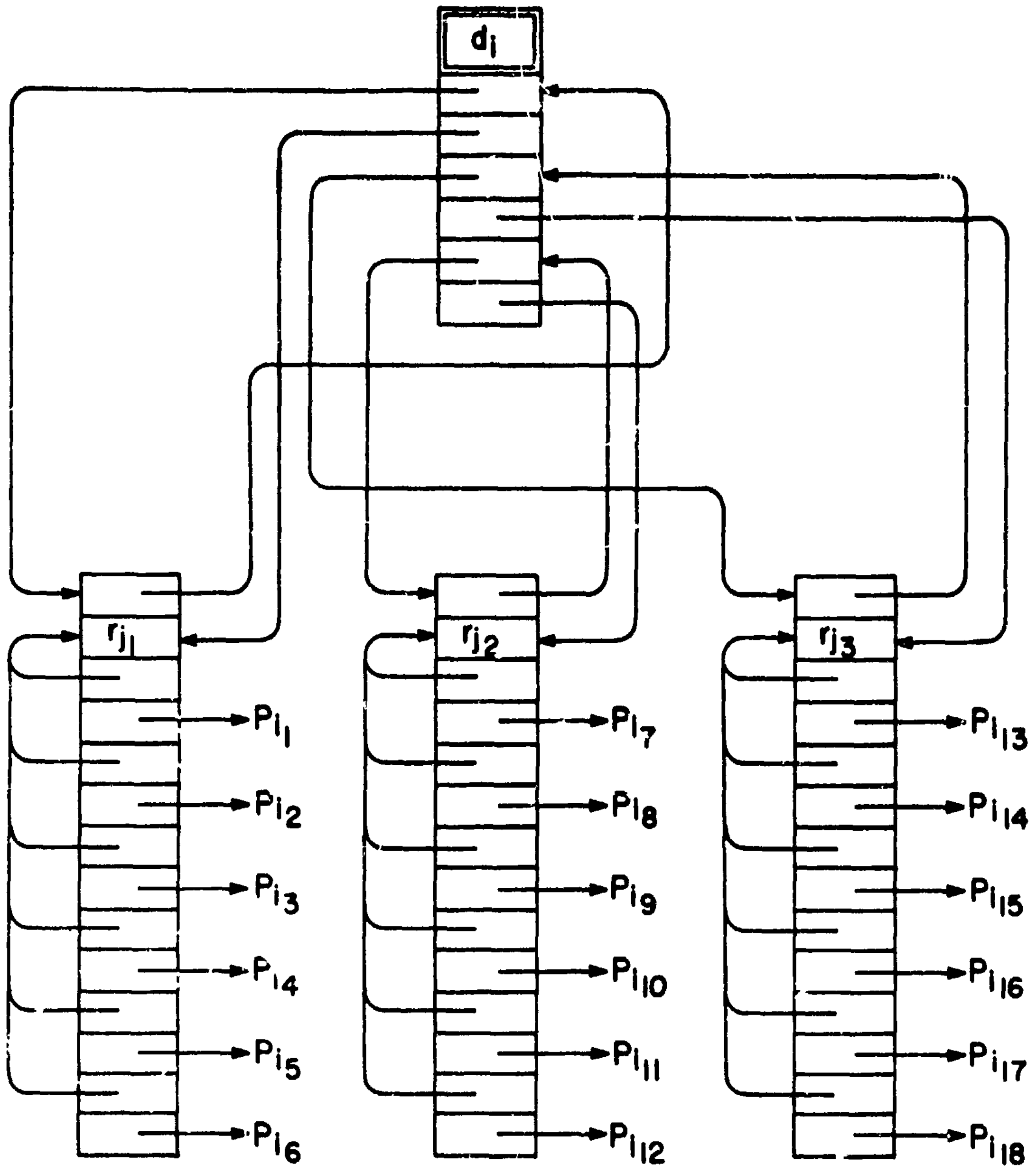


Figure 6-3. Optimal Storage Structure for Case 1

Taking omission of the relation rings and the indicated source ring pointers into account results in the reduction of the storage cost of the storage structure to 31414 units.

Now let us examine Solution 2 of ϕ-family 28. This solution has the schematic representation of Figure 6-4. We note that this schematic is the same as that of Figure 6-2 with the exception that each $b_6$-block of Figure 6-2 is replaced by a $b_6$-block - $b_8$-block pair and each $b_7$-block is replaced by a $b_9$-block.

As a result, the physical structure of this solution is identical to that of the first solution considered.

In fact, this is true for the four solutions of ϕ-family 10, also!

Thus, although each of the six given solutions has a unique specification, they all result in the same physical storage structure.

## 6.1.7 Justification of the Solution

We will now present an intuitive justification for the solution we have obtained. Because of their relative weights, operations $Q_3$ and $Q_7$ (especially operation $Q_7$) would be expected to influence the form of the structure considerably more than operations $Q_{12}$ and $Q_{14}$.

Of the three alternative methods available for operation $Q_3$ - namely, $\hat{d}\,\hat{r}\,p$, $\hat{r}\,\hat{d}\,p$, and $p\,\hat{r}\,\hat{d}$ - we would expect either of the first two to be preferable to the third on the grounds that fewer possible

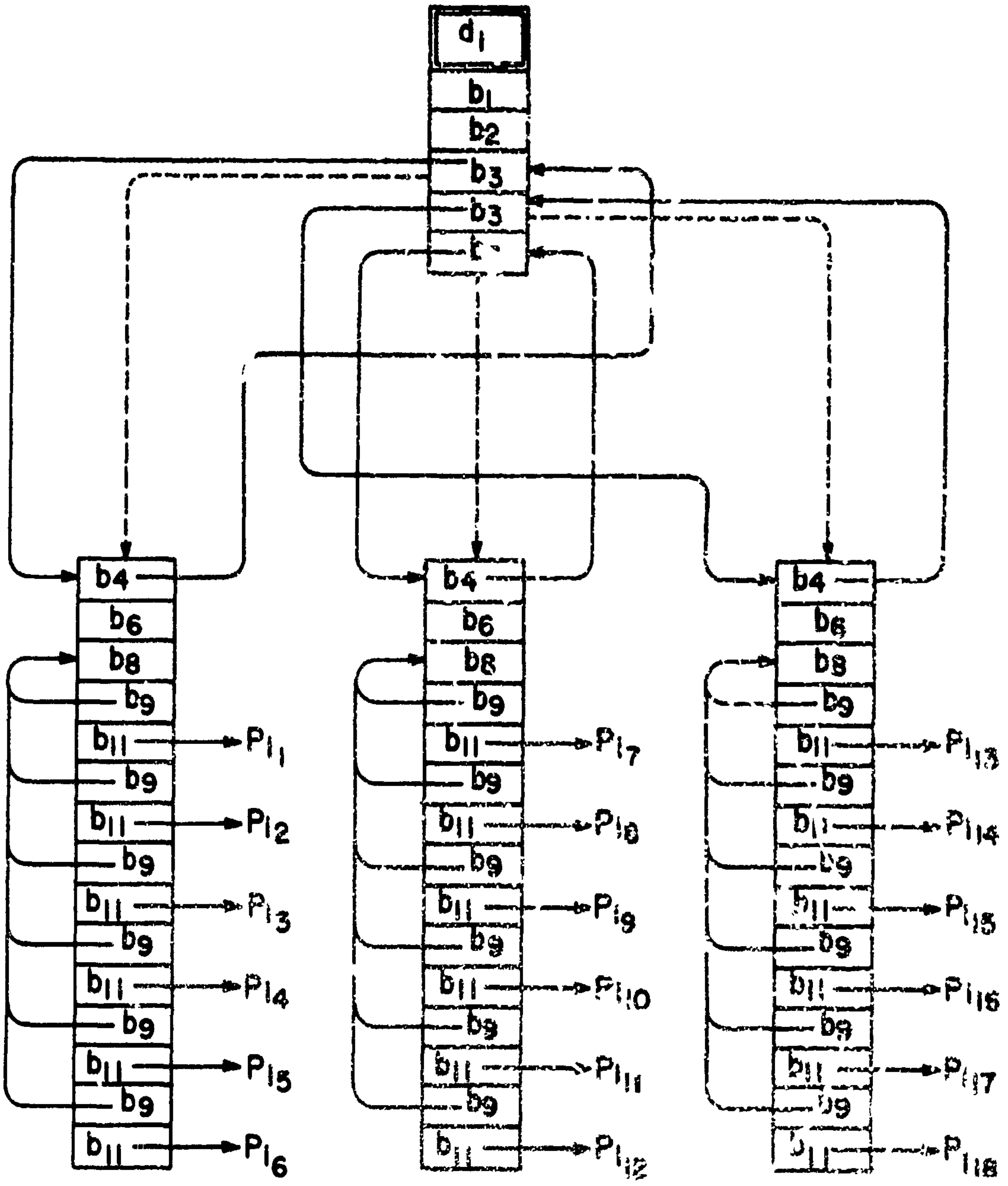


Figure 6-4. Schematic of Solution 2 for Case 1

solutions must be considered when using one of them than when using the third. By the same token, since the number of distinct relations which apply to a given source is less than the number of sources associated with a given relation, we would expect the first method to be preferable to the second.

Similarly, of the four methods available for operation $Q_7$ - $\hat{d}\ r\ \hat{p}$, $\hat{p}\ r\ \hat{d}$, $r\ \hat{d}\ \hat{p}$, and $r\ \hat{p}\ \hat{d}$ - we would expect the first two to be preferable to the last two. If operation $Q_7$ were considered by itself, either of the first two methods would probably serve equally well. However, since our choice of methods for operation $Q_3$ favors starting with sources, we are encouraged to choose the first method for operation $Q_7$.

In light of our choices of methods for operation $Q_3$ and $Q_7$, we would certainly prefer method $\hat{d}\ p$ to method $p\ \hat{d}$ for operation $Q_{12}$.

In view of our choices so far we might be tempted to choose method $d\ \hat{p}$ over method $\hat{p}\ d$ for operation $Q_{14}$, but the latter is probably preferable in all but the most extreme cases and so we choose it. In any event since the relative weight of operation $Q_{14}$ is so small, our decision here probably will not have a very great effect upon the form of the storage structure anyway. (We note that it is responsible for inclusion of the $a_6$-ring head pointers, however.)

Solution of our problem does, of course, support our intuitive choices of methods.

Let us examine the structure of Figure 6-3 given our choice of methods for the various operations. In order for operation $Q_3$ to be as efficient as possible, the relations which apply to a given source should be easily accessible from that source. At the same time to promote efficiency for operation $Q_7$, the targets (as well as the relations) associated with a given source should be easily accessible.

Intuitively one might feel that duplication of the $b_4$-blocks upon the $b_3$-blocks should render operation $Q_7$ somewhat more efficient than the rings of our given storage structure. In this case access to all targets could be made by stepping through a single stack In the context of our current problem there are two pitfalls to this approach: First, the performance of operation $Q_3$ suffers in this case because we must step through a stack of description block indicators in order to access one relation symbol name field from another. Second, for operation $Q_7$ itself we must step all the way through a stack of description block indicators (to reach the next relation symbol) even though the target of interest may already have been found. With the structure of Figure 6-3 this is not necessary. Thus, the rings of our structure would appear to be the logical choice.

Once a given relation symbol has been found, a stack of description block indicators (as contained in the given structure) would appear to be the best choice for both operations $Q_3$ and $Q_7$.

Because of the similarlity between operation $Q_{12}$ and operations $Q_3$ and $Q_7$ and because of the small relative weight of operation $Q_{12}$, the given structure should render it relatively efficient, also.

In spite of the fact that there are on the average 18 $b_{11}$-blocks for each target in our structure, operation $Q_{14}$ is still relatively efficient. The head pointers from the $b_7$-blocks to the $b_6$-blocks and the explicit $a_3$-rings both contribute to making access of the sources associated with a given target very fast. We note that duplication of the $b_4$-blocks upon the $b_3$-blocks would be a detriment to this operation, also.

Thus, the storage structure of Figure 6-3 is appealing even on an intuitive basis.

### 6.1.8 Comparison with an Existing Storage Structure

In order to put the solution we have obtained into proper perspective let us compare it with the scheme used by Gorry in his implementation of the diagnostic system.

Gorry's storage structure consists of a number of SLIP lists which are maintained and interrogated through use of the SLIP language. Although he concedes that this structure may at times result in inefficient utilization of the computer memory, he maintains that this disadvantage was more than offset by the convenience of programming in the high-level SLIP language. Since he was concerned with implementing a prototype system, we must agree wholeheartedly with this philosophy.

In his structure Gorry defines a basic information block (our data item) to be either a state (i.e., a disease), an attribute (i.e., a symptom), or a test. Each of these basic blocks is then represented by a SLIP list. The result is a number of <u>state lists</u>, <u>attribute lists</u>, and <u>test lists</u>.

A state list, as shown in Figure 6-5, consists of a ring of attribute list names (i.e., pointers to the heads of certain attribute lists), one for every attribute relevant to the state, and a description list (or DLIST) which contains the a priori probability of the given state and the print name of the state.

An attribute list, as shown in Figure 6-6, consists of a ring of test list names corresponding to those tests which can result in the given attribute and a DLIST which contains the print name of the attribute and a pointer to a <u>member list</u>. The member list contains the list name of each state list on which the name of the attribute list appears and the corresponding probability of the attribute given the state.

Finally, a test list, which appears in Figure 6-7, contains the cost

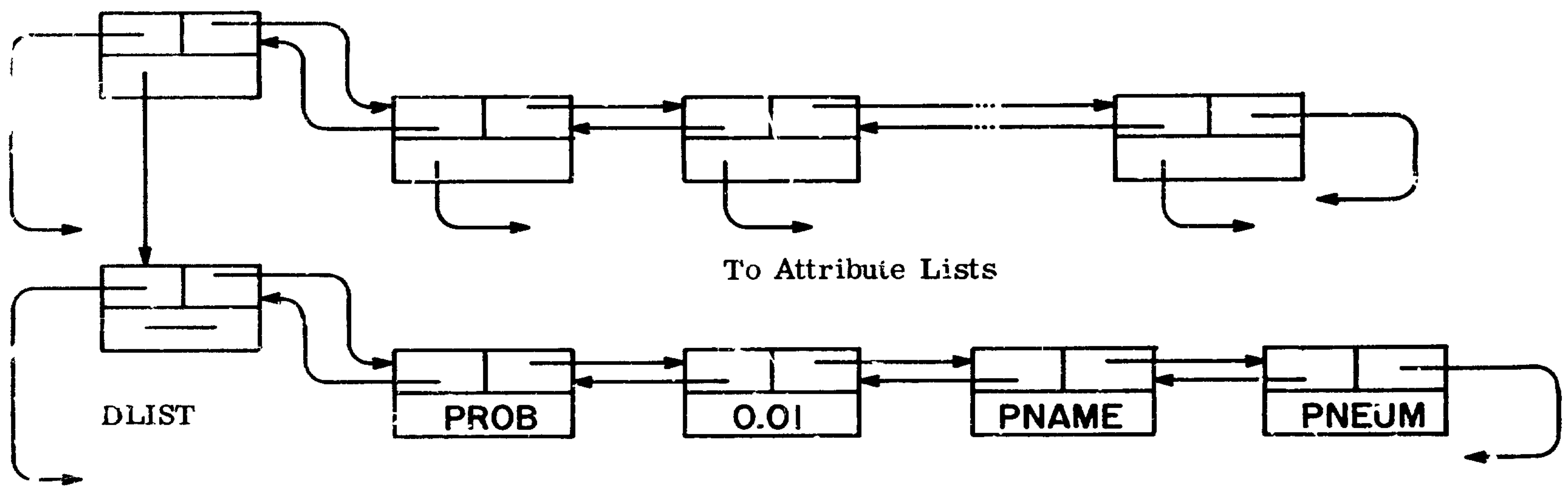


Figure 6-5. State List

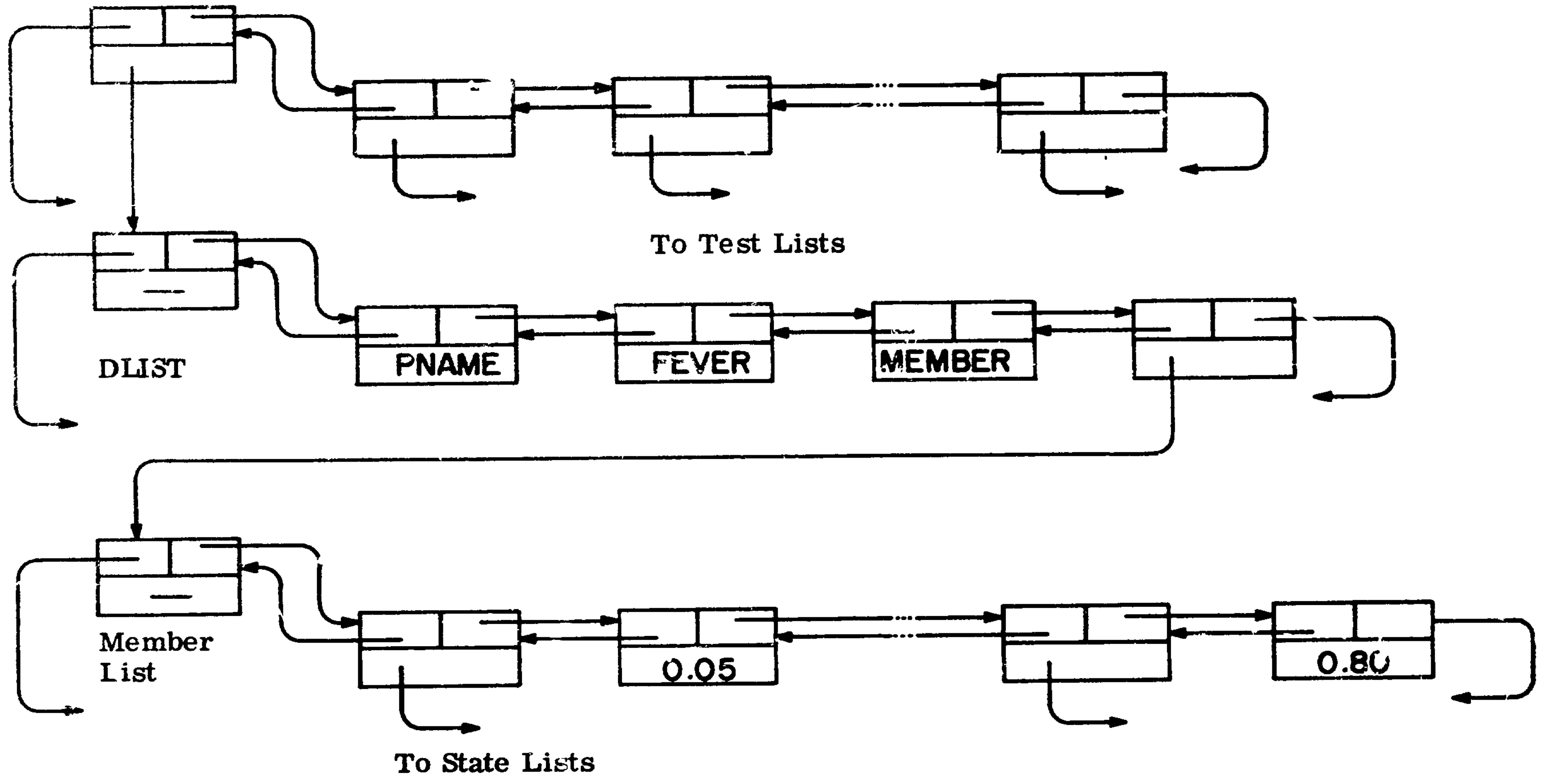


Figure 6-6. Attribute List

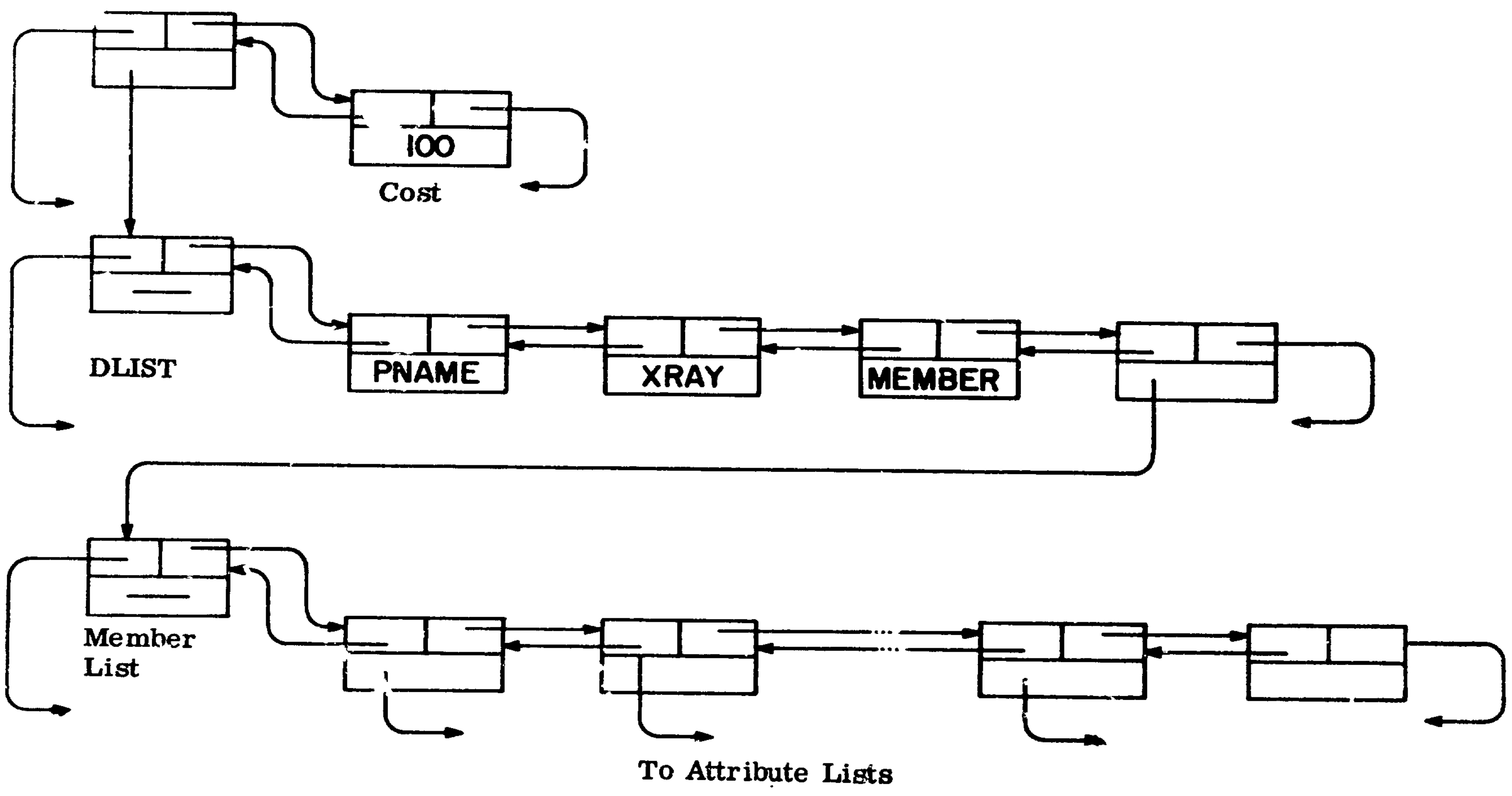


Figure 6-7. Test Lis'

of the test and a DLIST. The DLIST contains the print name of the test and a member list for the attribute lists which include this test.

For the heart disease problem which we have been considering each state list will contain an average of 28.4 attribute list names; each attribute list will contain 1 test list name and its member list will contain an average of 18.7 pairs of state list names and probabilities; and each test list will have an average of 1.7 attribute list names in its member list.

Using this information and the description of the environment used in determining our optimal storage structure, let us determine the cost of each of the basic operations involved in the diagnostic process. To reiterate briefly, the basic operations are: (1) determine the diseases associated with a given symptom, (2) determine the symptoms associated with a given disease, (3) determine the conditional probability that a particular symptom is associated with a given disease, (4) determine the tests relevant to a given disease, and (5) determine the possible outcomes of a given test.

The first of these operations involves finding the attribute list corresponding to the given symptom and recording the list name of each state list on its member list. (The print name of each disease is irrelevant at this point since the results of this operation are simply used as the starting points for the second operation.) Finding the head of the attribute list requires $T_a+F_a$ units of time . Accessing the head of the corresponding member list involves moving to the DLIST at a cost of $s_o + f$ time units (where f represents the time cost to follow a pointer), stepping through the DLIST at a cost of 4f time units (ignoring the cost of any comparisons which must be made), and finally moving to the head of the member list at a cost of $s_o+ f$ time units. Recording the list name of each of the state lists then involves stepping through the member list at a cost of 18.7 (2f) and fetching each of the state list names at a cost of $s_o$ plus the actual cost of a fetch (which we will represent by v and assume requires 7 time units). Summing these incremental costs yields a time cost of 353.9 for this operation.

The second operation starts at the head of a state list and records the print name of each attribute on the list. The time cost for this operation is given by the expression

$$28.4\ (f + s_o + 4f + s_o + v)$$

where f time units are required to access an element of the state list (from the previous one), $s_o$ units are required to move to the corresponding attribute list name, 4f units are required to access the print name block on the DLIST of the given attribute list, and $s_o + v$ units are required to fetch the print name . As a result this operation has a time cost of 880.4 time units.

The third operation - determining the conditional probability of a symptom given a disease - involves finding the attribute list corresponding to the given symptom, searching the associated member list for the given disease, and recording the corresponding probability. To the point of accessing the member list this operation is identical to the first operation and, hence, requires

$$T_a + F_a + s_o + s_f + s_o + f$$

time units to reach this point in the structure. On the average we would expect to examine (18.7 + 1)/2 state list entries of the member list before finding the given disease. For each one examined we must access the member list element block (from the previous one) at a cost of f time units, access the head of the corresponding state list at a cost of $s_o + f$, then access and compare the print name at a cost of $5f + s_o + c$ (where c represents the cost of a comparison and has the value 8), and finally move to the member list element block containing the value of the probability. For the last state list considered we must fetch the value of the probability, which costs $s_o + v$ time units. The resultant time cost for this operation is 437.0 time units.

To perform the fourth operation - determination of the tests relevant to a given disease - we must access the state list representing the disease and access in turn the associated attribute lists, which contain the list names of the pertinent tests. This requires $T_a + F_a$ time units to access the head of the state list, $f + s_o + f$ units to access each of the 28.4 attribute lists, $f + s_o + f$ units to access the corresponding test list, and $3f + s_o + v$ time units to fetch the print name of the test. The time cost for the operation is therefore 1172.4 time units.

Finally, the fifth operation - determining the possible outcomes of a test - involves fetching the print names of the attribute associated with the given test list. To do this we must access the member list for the given test at a cost of $T_a + F_a + 5f + s_o + f$ and access the print name of each of the 1.7 attribute lists thereon at a cost of $1.7\ (f + s_o + 4f + s_o + v)$. The resultant time cost is 86.7 time units.

In order to make a fairly coarse comparison of the SLIP structure with the optimal structure, let us compare these time costs with those determined for the primitive operations. Operations 1, 2, and 3 correspond to primitive operations $Q_{12}$, $Q_{14}$, and $Q_7$, respectively, and operations 4 and 5 correspond to primitive operation $Q_3$. Table 6-10 contains a summary of the time costs for both sets of operations, where the costs of operations 4 and 5 have been weighted according to their relative frequencies (operation 5 is used five times as much as operation 4) and combined to facilitate comparison with the time cost for primitive operation $Q_3$. The corresponding values of the weighted

| SLIP Structure | | | Optimal Structure | |
|---|---|---|---|---|
| Operation | Cost | Relative Frequency | Operation | Cost |
| 1 | 353.9 | 0.001 | $Q_{12}$ | 272.00 |
| 2 | 880.4 | 0.001 | $Q_{14}$ | 530.00 |
| 3 | 437.0 | 0.806 | $Q_7$ | 285.22 |
| 4 & 5 | 267.6 | 0.192 | $Q_3$ | 116.00 |

T = 404.8 T = 252.96

Table 6-10. Operation Time Costs for the SLIP Structure versus the Optimal Structure

time cost function also appear in Table 6-10. We see that the time cost T for the SLIP structure is 60 percent greater than that for the optimal structure.

In order to achieve a somewhat better comparison of the two structures let us determine the cost of each of the five basic operations using the optimal storage structure.

There are two reasons why computing the costs of the five basic operations for the optimal structure (instead of using the costs determined for the primitive operations in obtaining the optimal structure) should give us a better basis for comparison.

First, in certain cases our primitive operations are only approximations to the corresponding basic operations which they ostensibly represent. For example, we note that primitive operation $Q_7$ is really more general than operation 3, its counterpart. In particular, $Q_7$ examines _all_ relations associated with a given symptom (acting as a source) to determine whether or not the given disease is a possible target. In practice we know that there is only one such relation and once it has been found all others may be disregarded. This mismatch of operations could be easily overcome, of course, by introducing another primitive operation which searches for exactly one relation (possibly the first encountered out of many, or else the only one) which associates a given target with a given source. Adopting such a posture for all similar cases might result in a plethora of primitive operations, however. In addition, the added effort required to derive time cost expressions for these additional primitive operations might not be rewarded by proportionate increases in the accuracy of the optimal structure obtained.

The second reason for computing the basic operation costs for the optimal structure is that each of these time costs may reflect the actual (average) numbers of blocks we may expect to encounter in the structure for the relations with which the corresponding operation is intimately concerned. The primitive operation time costs which we are given, on the other hand, reflect the average over all relations.

Let us proceed, therefore, with calculations of the time costs for the five basic diagnosis operations, using the optimal storage structure.

The first operation - determining the diseases associated with a given symptom - involves finding all the description blocks for diseases associated with a given symptom. To perform this operation we utilize that portion of the structure representing the relations $r^o_1 \cdots r^o_{20}$, which have symptoms as sources and diseases as targets. We must find the description block representing the given symptom, step through a stack of on the average 6.6 $a_3$-rings (using the notation of Solution 1 for ∅-family 28), and for each of these step through a stack of 2.8 $b_{11}$-blocks to access a total of 18.7 description block indicators.

Accessing the symptom description block requires $T_a + F_a$ time units, stepping through the $a_3$-rings and following their respective head pointers to the stacks of description block indicators requires 6.6 $(s_o + f)$ time units, and recording the various disease description block indicators requires 18.7 $(s_o + v)$ time units. The resultant time cost is 215.9 time units.

The second operation - determining the symptoms associated with a given disease - also relies upon that portion of the storage structure which represents relations $r^o_1 \cdots r^o_{20}$. For this operation, however, we must access (via the appropriate target ring) all $b_{11}$-blocks representing a given disease and move upward through the structure to access the corresponding symptom description blocks. For this case $m_p$ will have an average value of 28.4. Therefore, accessing all $b_{11}$-blocks representing a given disease will require $T_a + 29.4F_a$ time units. Since the stack of $b_3$-blocks contains on the average 6.6 blocks, we would expect to step through (6.6+1)/2 of these on the way to the symptom description block*. Thus, for each $b_{11}$-block encountered the time cost required to access the corresponding symptom description block will be $s_o + f + s_o + f + 3.8s_o$. The total time required for this operation will then be 678.2 time units.

* Note that if the value of $k^o_4$ had actually been 6.6 instead of 3 the optimal storage structure would have contained head pointers to facilitate this operation.

The third operation - determination of the conditional probability that a symptom is associated with a given disease - is very similar in implementation to the first operation. In this case, however, since we may halt our transversal of the structure once we have found the particular disease sought, we need consider an average of only (6.6+1)/2 $a_3$-rings and (18.7+1)/2 disease description blocks. Thus, accessing the symptom description block requires $T_a + F_a$ time units, stepping through the stack of $a_3$-rings and following their respective head pointers requires $3.8(s_0 + f)$ time units, examining the various disease description blocks requires $9.8\ (s_0 + f + c)$ time units, and fetching the value of the probability relating the given symptom and the given disease requires v time units. The time cost of the third operation is therefore 175.0 time units.

Operation 4 - determining the tests relevant to a given disease - uses the portion of the storage structure which represents the relation $r_1$, which has diseases as sources and their relevant tests as targets. This operation involves accessing a disease description block at a cost of $T_a + F_a$ time units, stepping through a stack containing a single $a_3$-ring and accessing the corresponding stack of $b_{11}$-blocks at a cost of $s_0 + f$, and fetching the tests indicated by the test description block idicators in this stack (19.3 of them on the average) at a cost of 19.3 $(s_0 + f + v)$. This results in a time cost of 264.9 time units.

Operation 5 - determining the possible outcomes of a given test - uses that part of the structure representing the relation $r_2$, which has tests as sources and symptoms as targets. Its implementation is identical to that for the fourth operation, but since the stack of $b_{11}$-blocks contains on the average only 1.7 blocks, its time cost is 36.1 time units.

Table 6-11 contains a summary of these time costs and the overall time cost in comparison with the corresponding quantities for the SLIP structure. On the basis of this comparison we see that the time cost T for the SLIP structure is 2.6 times greater than that for the optimal structure.

Let us also compare the storage costs of the two structures under consideration. Assuming that each field of a SLIP list block requires 4 bytes of storage, each block then requires 12 bytes. Summing the number of blocks required to implement a list over 33 state lists, 50 attribute lists, and 30 test lists results in a total storage cost of 44355 units, which is 41 percent greater than the 31414 units required by the optimal storage structure.

Finally, let us investigate how the SLIP structure might be described by our storage structure model, and then determine the time and storage costs for the diagnostic problem using this structure and the required primitive operations.

| Operation | Time Cost | |
|---|---|---|
| | SLIP Structure | Optimal Structure |
| 1 | 353.9 | 215.9 |
| 2 | 880.4 | 678.2 |
| 3 | 437.0 | 175.0 |
| 4 | 1172.4 | 264.9 |
| 5 | 86.7 | 36.1 |
| | T = 404.8 | T = 156.2 |

Table 6-11. Basic Operation Time Costs for the SLIP Structure Versus the Optimal Structure

If we examine Figures 6-5, 6-6, and 6-7 closely, we see that the state, attribute, and test lists are all basically the structure in Figure 6-8.

In particular, for the state list if we assume that the head of the list and the DLIST may be combined into the description block (the double-walled block) of Figure 6-8, then except for the presence of relation symbol name fields the remainder of the structure is identical to that of Figure 6-8.

If we ignore for the moment the associated member list, then the same can be said for the attribute list. If we now combine each pair of blocks in the member list into a single block like those in Figure 6-8 and if we assume that the head of the member list and the DLIST are combined, we have for the member list exactly the structure of Figure 6-8. Thus, the attribute list can be considered to consist of a single description block with two associated rings like that of Figure 6-8.

If the test list itself, its DLIST, and the head of its associated member list are combined into a single block, again except for the relation symbol name fields we have the structure of Figure 6-8.

Since each of the state lists, attribute lists, and test list member lists may be viewed as representing all the relation instances having a given source for a single particular relation, we may use the

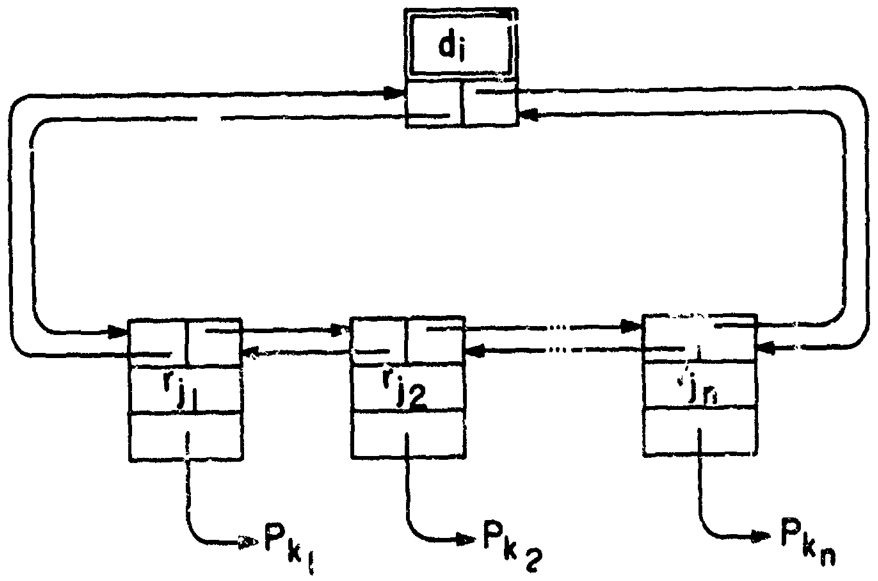


Figure 6-8. Basic SLIP Structure

structure of Figure 6-8 (including the relation symbol name fields) to represent each of these lists and assume that for each list all of the relation symbol name fields contain the same relation symbol name and occupy zero storage units. For the attribute list member list the relation symbol name fields may contain the various conditional probabilities in a situation analogous to the representation of our relations $r_1^o \cdots r_{20}^o$. In speaking of a single homogeneous structure for the representation of all the lists we may then use the structure of Figure 6-8 and specify the size of the relation symbol name field as being the average of those for the various lists (i.e., $s_r = 2$ bytes).

It should be clear that within the context of the SSM the structure of Figure 6-8 may have the description contained in Table 6-12, where we have ignored the presence of reverse pointers.

Before determining values for the unspecified parameters $k_4^o$, $k_8^o$, and $k_9^o$ let us determine the values of $k_a^o$, $k_p^o$, and $k_r^o$. We would normally say that the number of sources is equal to the number of states plus the number of attributes plus the number of tests (i.e., $k_a^o = 113$). However, since we wish to associate two rings with each attribute, we see that the attributes have two distinct roles in the structure and as a result we will count each one twice so that $k_a^o = 163$. (We might have two source rings associated with each attribute, but only a single description block.) We will count each attribute as a target only once

| | | |
|---|---|---|
| $\phi_1 \cdots \phi_{10}$: | 0 0 0 1 0 0 0 0 0 0 | |
| $\phi'_1 \cdots \phi'_{10}$: | 0 0 0 0 0 0 0 0 0 0 | |
| $\Delta_1 \cdots \Delta_{10}$: | 1 1 1 0 1 1 1 1 1 1 | |
| $\beta_2 \cdots \beta_{10}$: | 1 1 0 0 0 1 1 1 1 | |
| $\sigma_1 = 0$ | $\sigma_2 = 1$ | |
| $\rho_1 = 0$ | $\rho_2 = 1$ | $\rho_3 = 0$ |

$k^o_1 = 1$

$k^o_2 = 1$

$k^o_3 = 1$

$k^o_4 = ?$

$k^o_7 = 1$

$k^o_8 = ?$

$k^o_9 = ?$

$k^o_{10} = 1$

A question mark indicates an unspecified value greater than or equal to 1.

Table 6-12. Specifications for the Basic SLIP Structure

since there is no dichotomy of roles here, so $k_p^o = 113$. Finally, $k_r^o = 23$, twenty relations corresponding to the distinct probabilities and one each for the state list, attribute list, and test member list functions.

Let us now consider $k_4^o$, $k_8^o$, and $k_9^o$. $k_4^o$ corresponds to the average number of relation symbols for each of the state lists, attribute lists, attribute member lists, and test member lists. We can easily determine, therefore, that $k_4^o = 2.7$.

$k_8^o$ represents the average number of targets for each source-relation symbol pair and, hence, will have the value $k_8^o = 4.4$

Since the expression

$$k_2^o k_4^o k_8^o k_{10}^o k_a^o = k_1^o k_3^o k_7^o k_9^o k_p^o$$

must be satisfied, it follows that

$$k_9^o = k_4^o k_8^o k_a^o / k_p^o$$

which yields the result $k_9^o = 17.1$.

Because of the constraint imposed by our program requiring these parameters to be integer valued, let us assign the following values to them:

| | | |
|---|---|---|
| $k_a^o = 150$ | $k_p^o = 100$ | $k_r^o = 23$ |
| $k_4^o = 3$ | $k_8^o = 4$ | $k_9^o = 18$ |

Finally, let us assume $m_{r_p} = 4$. (This should be approximately the same as the value derived for $m_{r_p}$ in specifying the values of parameters for our derivation of the optimal structure and must be integer valued, also, so we choose the value 4.)

As a last step before computing the measures of performance for this storage structure we must determine the primitive operations to be used and their relative frequencies. Basic operation 1 corresponds to applying primitive operation $Q_{12}$ to the member list associated with a given attribute. Operation 2 may be accomplished by applying primitive operation $Q_3$ to the proper state list. Operation 3 may be represented by primitive operation $Q_7$ applied to an attribute member list. Operation 4 involves applying primitive operation $Q_3$ recursively - first to a given state list and then to each of the attribute lists indicated by the first application. Lastly, operation 5 corresponds to primitive operation $Q_3$ as applied to the proper test member list. It follows that the relative frequencies of primitive operations $Q_3$, $Q_7$, and $Q_{12}$ are as given below:

| Operation | Frequency | Relative Frequency |
|---|---|---|
| $Q_3(d_i r_j -)$ | 35095 | 0.570 |
| $Q_7(d_i - p_k)$ | 26396 | 0.429 |
| $Q_{12}(d_i * -)$ | 13 | 0.001 |
| | 61504 | |

If we now use our program to compute the time and storage costs for the basic SLIP structure under consideration (where the environment is assumed the same as that used in determining our optimal structure), we find that

$$T = 248.00$$

$$S = 35684$$

At first glance it might appear as though this structure is superior to our so-called optimal structure! Indeed, it is true that the average time cost for the performance of a single operation (which is essentially what the time cost function T represents) for this structure is less than that for our optimal structure (comparing the two computer determined time costs). It is also true, however, that more operations must be performed for this structure than for the optimal structure in order to achieve the same result. In particular we see from our calculations of the relative frequencies of the primitive operations applied to each of the storage structures that whereas the optimal structure requires the performance of 32734 operations for the "solution" of the diagnostic problem, the basic SLIP structure requires the performance of 61504 operations. If we normalize the time cost T for the SLIP structure to 32734 operations (i.e., if we multiply this time cost by the ratio 61504/32734) we find that the equivalent time cost is 466.24. Thus, the time cost for the basic SLIP structure of Figure 6-8 is really 1.8 times that of the optimal storage structure when compared with the program determined time cost (252.96) for the optimal structure.

To complete this comparison we might correct the storage cost given above to account for the additional attribute description blocks and the missing reverse pointers. To wit, S = 43184 storage units, which is 37 percent greater than that required by the optimal structure.

We might also find it instructive to compare the time and storage costs for the basic SLIP structure with those for Gorry's SLIP structure. There should be a high correlation between the values obtained for the two structures.

If we compare the normalized time cost for the basic SLIP structure as we have computed it here (466.24) with the time cost we have determined for Gorry's SLIP structure (404.8), we see that the time cost for the basic SLIP structure is 15 percent greater than that for Gorry's structure. This difference can probably be attributed to the fact that primitive operation $Q_7$ is more general than its counterpart, operation 3. (We discussed this matter somewhat earlier.)

If we now compare the storage cost for the basic SLIP structure (43184) with the for Gorry's SLIP structure (44355), we find that the basic structure has a storage cost within 3 percent of that for Gorry's structure.

These results would tend to imply that even though our model uses average values to characterize a particular data structure and its associated storage structure, the model may be used to determine a good estimate of the time and storage costs for existing or proposed structures, in addition to determining the optimal storage structure for a given application.

The conclusion which we wish to draw from the foregoing comparisons is quite simple: Gorry's SLIP storage structure, although presumably designed to be relatively efficient in operation, can be significantly surpassed in performance. Perhaps we do Gorry an injustice by making this statement, for he does admit to possible inefficient utilization of memory (clearly, for this application the reverse pointers of the SLIP structure are not required). In any event, his structure gives us a reference point for determining the relative improvement in performance we might expect by using a systematic procedure and a rigorous framework in the design of data representation schemes.

## 6.2 Near-Optimum Solutions

Our next goal is to examine the relative sharpness of the valley in which the optimal solution resides in order to determine just how critical it is that we find the optimal solution. In other words we want to determine whether or not there are other solutions which in practice co'' se:ve as well as the optimum.

Figure 6-9 contains a plot of the time costs of the best solution for various ϕ-families . We see from this plot that there are a number of solutions which have time costs close to that of the optimal solution (ϕ-families 10 and 28). In particular, ϕ-families 4, 19, 37, 46, and 55 all contain solutions which have time costs within 3 percent of the time cost for the optimal solution*. This would tend to confirm the hypothesis that there is no absolute optimal solution, but rather a spectrum of optimal or nearly optimal solutions.

In order to obtain some feel for the "variance" we can expect from one solution to another in this spectrum of solutions, let us examine briefly the best solution for each of the ϕ-families indicated above. Descriptions of these solutions are contained in Table 6-13 in order of increasing time cost.

Upon close inspection of these various solutions we find in each case that the two solutions in a ϕ-family result in the same physical storage structure. Moreover, the solutions for ϕ-family 19 and ϕ-family 55 are identical. Figures 6-10 through 6-13 contain schematics for the solutions of ϕ-families 19 (and 55), 4, 37, and 46, respectively.

---

* The best solution for each of these ϕ-families has a time cost within 3 percent of that for the optimal solution. It is possible, of course, that these ϕ-families might contain other solutions within this range as well.

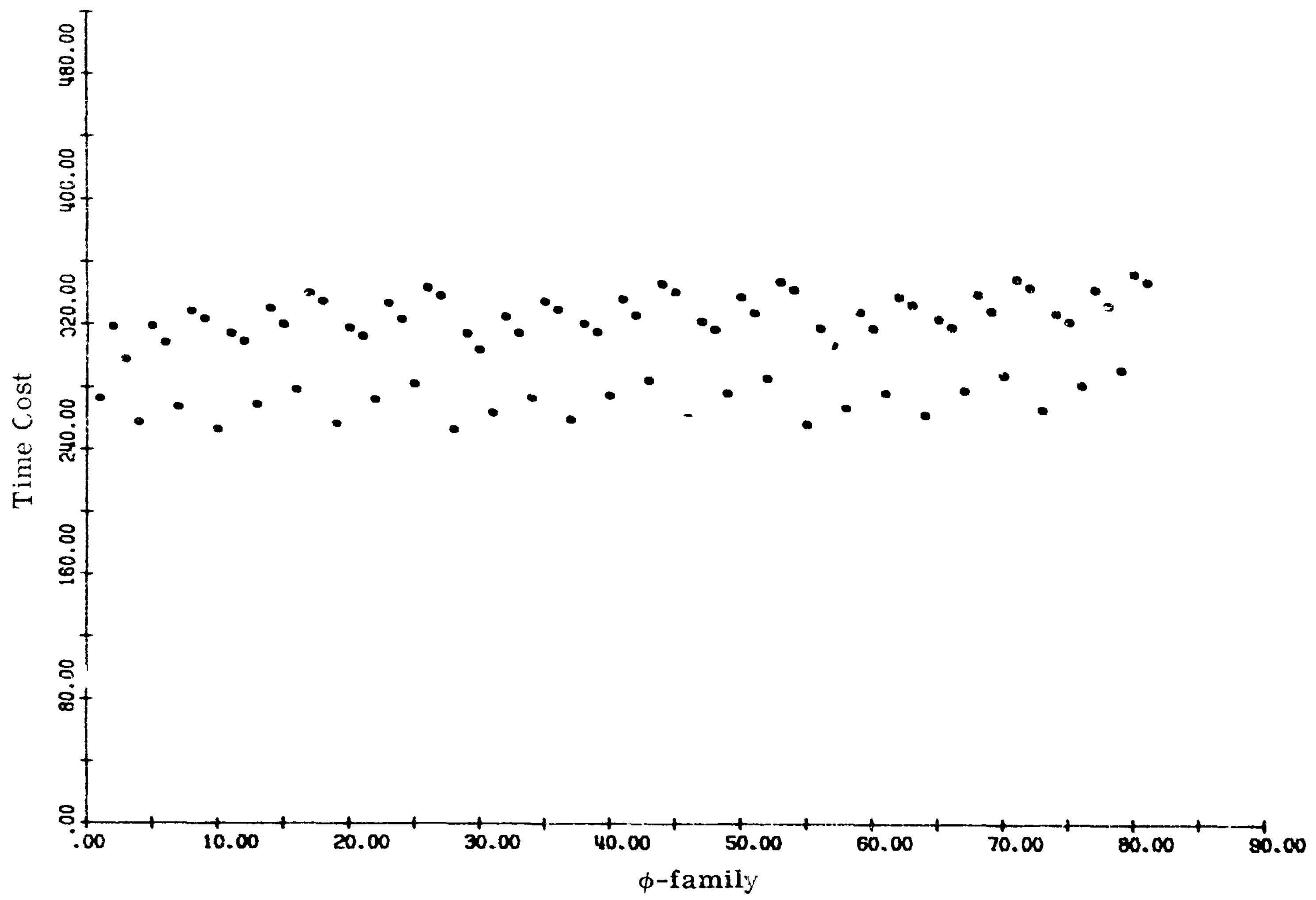


Figure 6-9. Time Costs by $\phi$-family for Case 1

$\phi$-family: 19

Solution 1

| | |
|---|---|
| $\phi_1 \cdots \phi_{10}$: | 0 0 0 1 0 0 0 0 0 0 |
| $\phi'_1 \cdots \phi'_{10}$: | 0 0 0 1 0 1 0 0 0 0 |
| $\Delta_1 \cdots \Delta_{10}$: | 1 1 1 0 1 0 1 1 1 1 |
| $\beta_2 \cdots \beta_{10}$: | 1 1 0 0 0 0 1 1 1 |
| $k_1 \cdots k_{10}$: | 1 1 1 3 1 6 1 1 1 1 |

Solution 2

| | |
|---|---|
| $\phi_1 \cdots \phi_{10}$: | 0 0 0 1 0 0 0 0 0 0 |
| $\phi'_1 \cdots \phi'_{10}$: | 0 0 0 1 0 0 0 1 0 0 |
| $\Delta_1 \cdots \Delta_{10}$: | 1 1 1 0 1 1 1 0 1 1 |
| $\beta_2 \cdots \beta_{10}$: | 1 1 0 0 0 1 0 0 1 |
| $k_1 \cdots k_{10}$: | 1 1 1 3 1 1 1 6 1 1 |

Common Items

$m_a = 1$ $m_p = 18$ $m_{r_1} = 1$ $m_{r_2} = 1$

$t_{3,1} = 118.00$

$t_{7,1} = 289.22$ $T = 256.42$

$t_{12,1} = 270.00$ $S = 32494$

$t_{14,1} = 386.00$

Table 6-13. Near Optimal Solutions for Case 1

**$\phi$-family: 55**

**Solution 1**

| | |
|---|---|
| $\phi_1 \cdots \phi_{10}$: | 0 1 0 0 0 0 0 0 0 0 |
| $\phi'_1 \cdots \phi'_{10}$: | 0 1 0 0 0 1 0 0 0 0 |
| $\Delta_1 \cdots \Delta_{10}$: | 1 0 1 1 1 0 1 1 1 1 |
| $\beta_2 \cdots \beta_{10}$: | 0 0 1 1 0 0 1 1 1 |
| $k_1 \cdots k_{10}$: | 1 3 1 1 1 6 1 1 1 1 |

**Solution 2**

| | |
|---|---|
| $\phi_1 \cdots \phi_{10}$: | 0 1 0 0 0 0 0 0 0 0 |
| $\phi'_1 \cdots \phi'_{10}$: | 0 1 0 0 0 0 0 1 0 0 |
| $\Delta_1 \cdots \Delta_{10}$: | 1 0 1 1 1 1 1 0 1 1 |
| $\beta_2 \cdots \beta_{10}$: | 0 0 1 1 0 1 0 0 1 |
| $k_1 \cdots k_{10}$: | 1 3 1 1 1 1 1 6 1 1 |

Common Items

$m_a = 1$ $\quad m_p = 18$ $\quad m_{r_1} = 1$ $\quad m_{r_2} = 1$

$t_{3,1} = 126.00$

$t_{7,1} = 289.22$ $\qquad T = 256.42$

$t_{12,1} = 270.00$ $\qquad S = 32494$

$t_{14,1} = 386.00$

Table 6-13. Near Optimal Solutions for Case 1 (Cont.)

$\phi$-family: 4

Solution 1

| | |
|---|---|
| $\phi_1 \cdots \phi_{10}$: | 0 0 0 0 0 0 1 0 0 0 |
| $\phi'_1 \cdots \phi'_{10}$: | 0 0 0 0 0 0 1 1 0 0 |
| $\Delta_1 \cdots \Delta_{10}$: | 1 0 1 1 1 1 0 0 1 1 |
| $\beta_2 \cdots \beta_{10}$: | 0 0 1 1 0 0 0 0 1 |
| $k_1 \cdots k_{10}$: | 1 3 1 1 1 1 1 6 1 1 |

Solution 2

| | |
|---|---|
| $\phi_1 \cdots \phi_{10}$: | 0 0 0 0 0 0 1 0 0 0 |
| $\phi'_1 \cdots \phi'_{10}$: | 0 0 0 0 0 0 1 1 0 0 |
| $\Delta_1 \cdots \Delta_{10}$: | 1 1 1 0 1 1 0 0 1 1 |
| $\beta_2 \cdots \beta_{10}$: | 1 1 0 0 0 0 0 0 1 |
| $k_1 \cdots k_{10}$: | 1 1 1 3 1 1 1 6 1 1 |

Common Items

$m_a = 1$ $m_p = 18$ $m_{r_1} = 1$ $m_{r_2} = 1$

$t_{3,1} = 114.00$

$t_{7,1} = 291.22$ $T = 257.41$

$t_{12,1} = 272.00$ $S = 33398$

$t_{14,1} = 530.00$

Table 6-13. Near Optimal Solutions for Case 1 (Cont.)

**$\phi$-family: 37**

Solution 1

| | |
|---|---|
| $\phi_1 \cdots \phi_{10}$: | 0 0 1 0 1 0 0 0 0 0 |
| $\phi'_1 \cdots \phi'_{10}$: | 0 0 1 0 1 1 0 0 0 0 |
| $\Delta_1 \cdots \Delta_{10}$: | 1 1 0 0 0 0 1 1 1 1 |
| $\beta_2 \cdots \beta_{10}$: | 1 0 0 0 0 0 1 1 1 |
| $k_1 \cdots k_{10}$: | 1 1 1 3 1 6 1 1 1 1 |

Solution 2

| | |
|---|---|
| $\phi_1 \cdots \phi_{10}$: | 0 0 1 0 1 0 0 0 0 0 |
| $\phi'_1 \cdots \phi'_{10}$: | 0 0 1 0 1 0 0 1 0 0 |
| $\Delta_1 \cdots \Delta_{10}$: | 1 1 0 0 0 1 1 0 1 1 |
| $\beta_2 \cdots \beta_{10}$: | 1 0 0 0 0 1 0 0 1 |
| $k_1 \cdots k_{10}$: | 1 1 1 3 1 1 1 6 1 1 |

Common Items

$m_a = 1$ $m_p = 18$ $m_{r_1} = 1$ $m_{r_2} = 1$

$t_{3,1} = 122.00$

$t_{7,1} = 291.22$ $T = 259.14$

$t_{12,1} = 278.00$ $S = 34754$

$t_{14,1} = 710.00$

Table 6-13. Near Optimal Solutions for Case 1 (Cont.)

$\phi$-family: 46

Solution 1

| | |
|---|---|
| $\phi_1 \cdots \phi_{10}$ | 0 0 1 1 0 0 0 0 0 0 |
| $\phi'_1 \cdots \phi'_{10}$: | 0 0 1 1 0 1 0 0 0 0 |
| $\Delta_1 \cdots \Delta_{10}$: | 1 1 0 0 1 0 1 1 1 1 |
| $\beta_2 \cdots \beta_{10}$: | 1 0 0 0 0 0 1 1 1 |
| $k_1 \cdots k_{10}$: | 1 1 1 3 1 6 1 1 1 1 |

Solution 2

| | |
|---|---|
| $\phi_1 \cdots \phi_{10}$: | 0 0 1 1 0 0 0 0 0 0 |
| $\phi'_1 \cdots \phi'_{10}$: | 0 0 1 1 0 0 0 1 0 0 |
| $\Delta_1 \cdots \Delta_{10}$: | 1 1 0 0 1 1 1 0 1 1 |
| $\beta_2 \cdots \beta_{10}$: | 1 0 0 0 0 1 0 0 1 |
| $k_1 \cdots k_{10}$: | 1 1 1 3 1 1 1 6 1 1 |

Common Items

$m_a = 1$ $m_p = 18$ $m_{r_1} = 1$ $m_{r_2} = 1$

$t_{3,1} = 122.00$

$t_{7,1} = 293.22$ $T = 260.60$

$t_{12,1} = 274.00$ $S = 33850$

$t_{14,1} = 566.00$

Table 6-13. Near Optimal Solutions for Case 1 (Cont.)

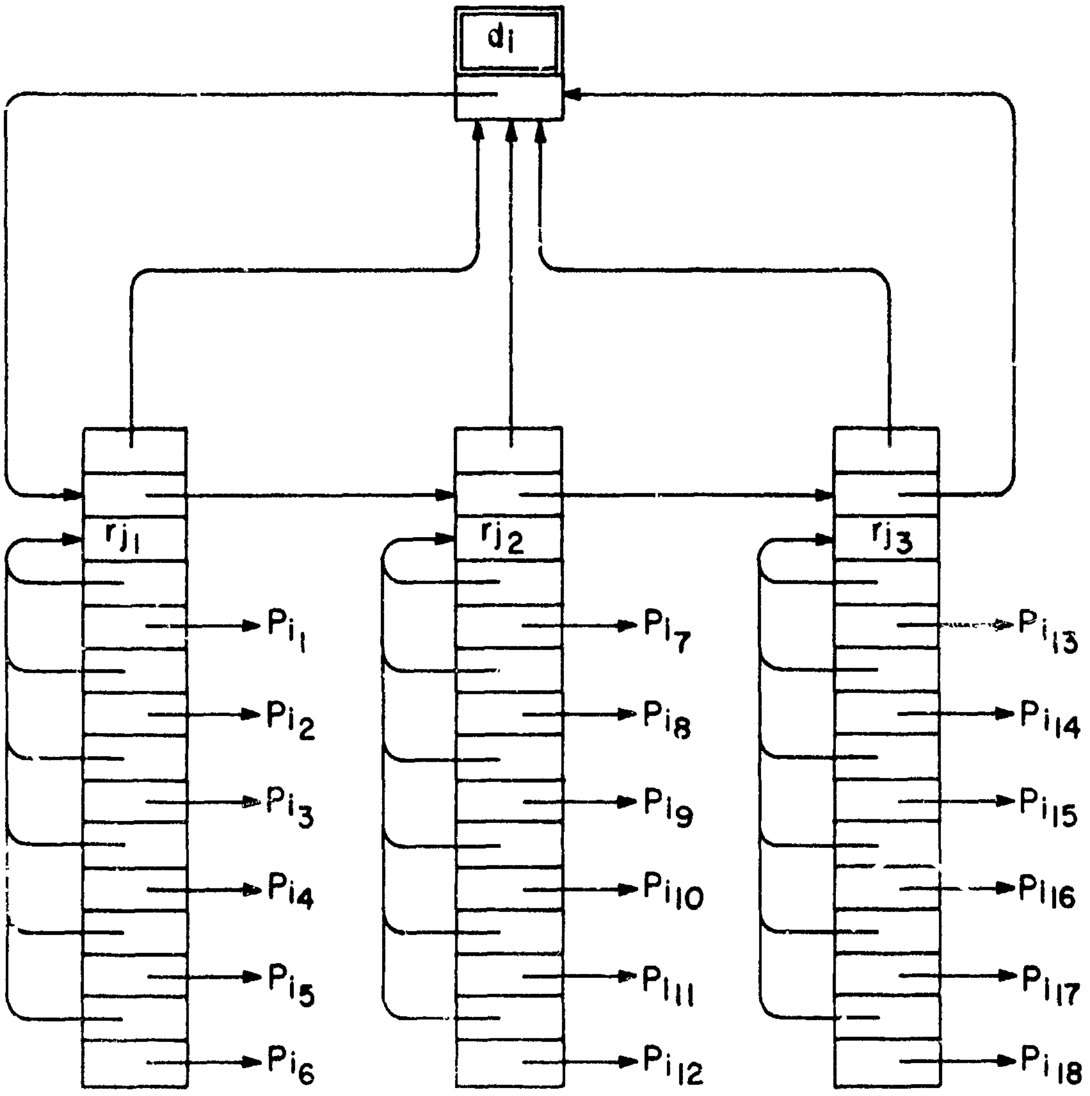


Figure 6-10. Best Solution for $\phi$-families 19 and 55 for Case 1

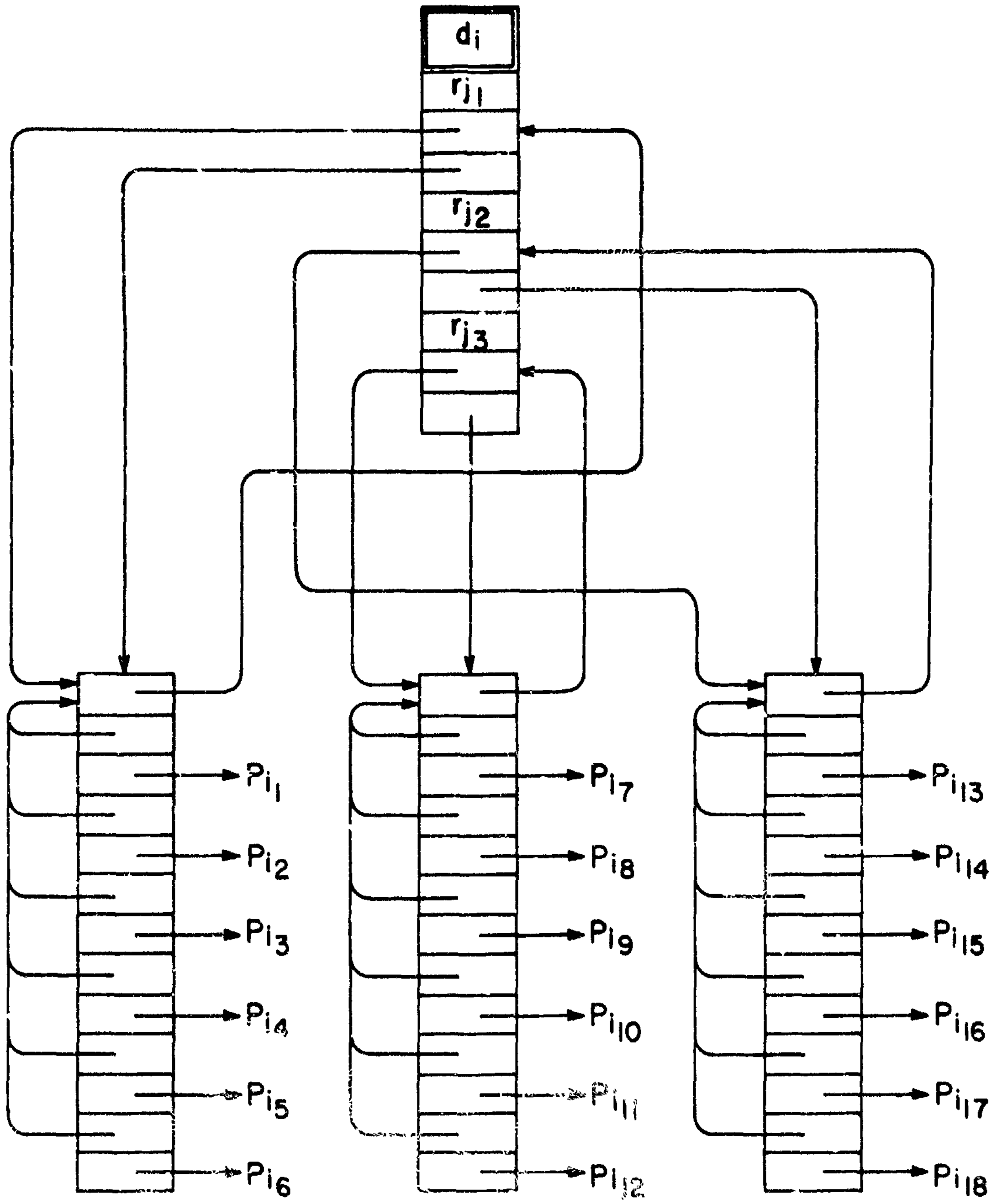


Figure 6-11. Best Solution for $\phi$-family 4 for Case 1

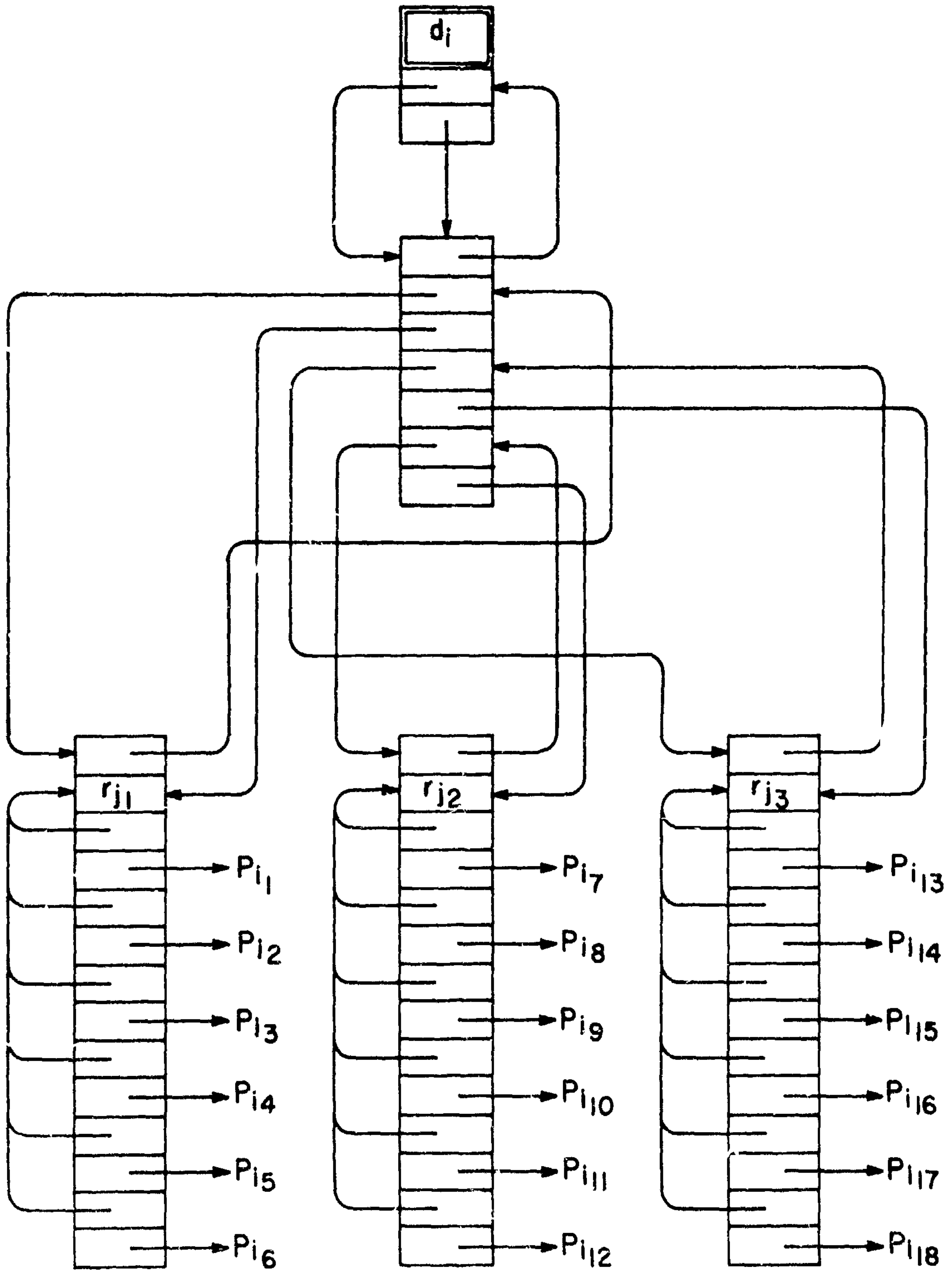


Figure 6-12. Best Solution for $\phi$-family 37 for Case 1

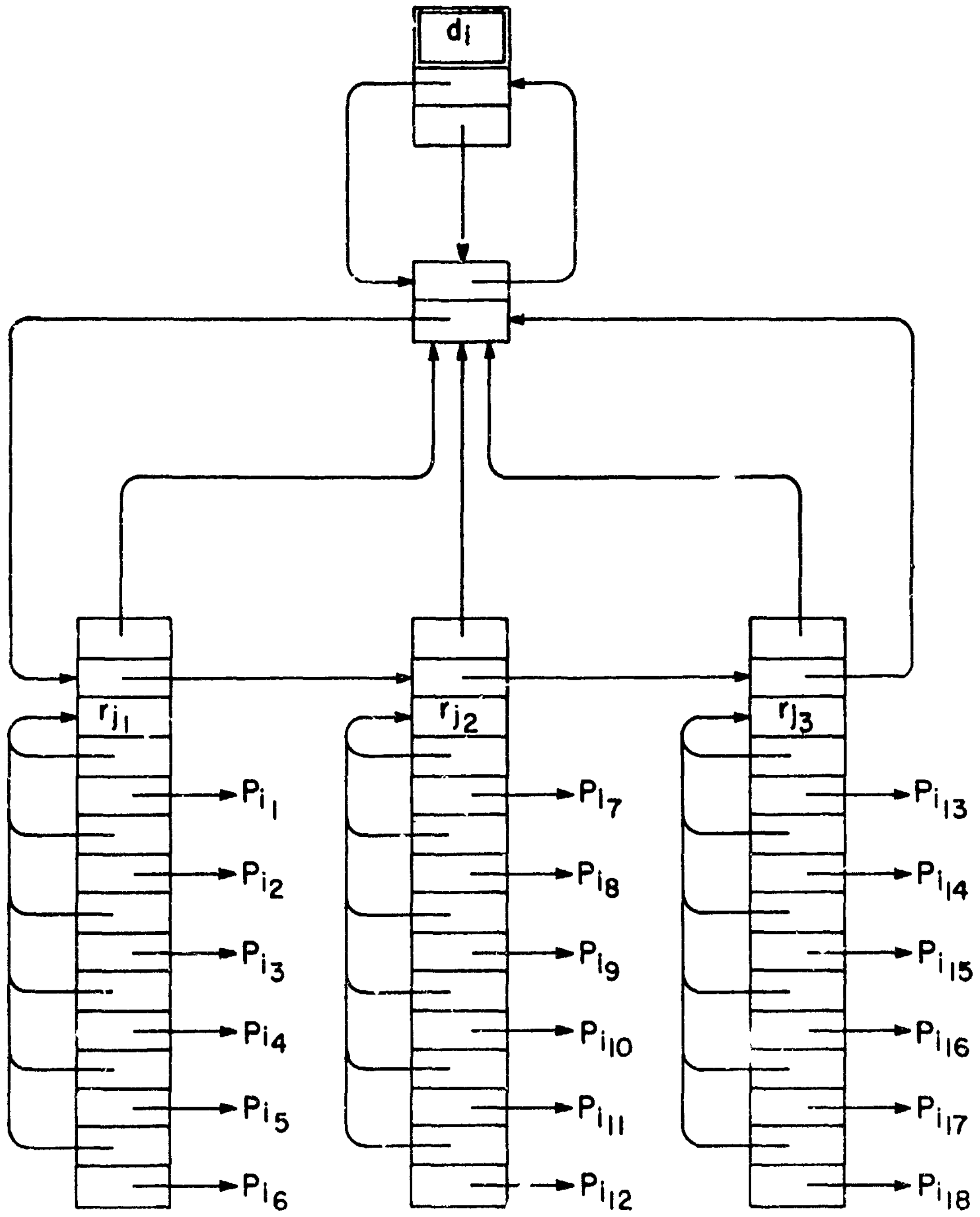


Figure 6-13. Best Solution for $\phi$-family 46 for Case 1

If we examine the structures of Figures 6-10 through 6-13 and compare them with the optimal structure of Figure 6-3, we can con clude that although there may be a number of optimal or nearly optimal solutions, all of these solutions tend to cluster about the optimal solution. That is, all of these structures are basically the same, differing only in relatively minor respects.

In particular, the storage structures which we have considered here all contain a stack of target description block indicators for each relation symbol associated with a given source. Furthermore, these stacks of description block indicators and their respective relation symbols are associated with a given source via a ring or stack of rings such that the overall storage structure takes the form of a downward branching tree (i.e., from source to target).

## 6.3 Sensitivity Analysis

Next we wish to determine the sensitivity of our optimal solution to variations in the problem under consideration. In particular, we wish to determine how sensitive our solution is to variations in $k_1^o \cdots k_{10}^o$ and to variations in the relative frequencies of the given prmitive operations. Since the values of these parameters must generally be determined by estimation, they are subject to the errors inherent in any estimation process. We would hope, therefore, that the solution is relatively insensitive to minor variations in these values.

### 6.3.1 Variation in $k_1^o \cdots k_{10}^o$

Let us first consider the parameters $k_1^o \cdots k_{10}^o$. Recall that the values used thus far in the solution of the diagnostic problem (i.e., in Case 1) are integer approximations to the values we determined for the heart disease problem. (See Tables 6-7 and 6-8.) Table 6-14 contains another equally valid set of values for $k_1^o \cdots k_{10}^o$ in approximation of the values given in Table 6-7. Let us now determine the optimal solution using the values given for $k_1^o \cdots k_{10}^o$ in Table 6-14 and keeping the values of all other parameters (including the external decision variables) unchanged.

Applying our program to these data, which we will designate Case 2, yields the solution listing contained in Appendix F. Figure 6-14 contains a plot of the time costs for the best solutions in each of the various ∅-families and Table 6-15 contains descriptions of the optimal and near optimal (within 4 percent of the optimum) solutions in order of increasing time cost. As was the case before, method 1 is the best choice for each of the primitive operations for all solutions.

As in the previous case, ∅-families 28 and 10 contain the optimal solutions and ∅-families 4, 19, 37, 46, and 55 contain the near optimal solutions with the same relative ordering. In addition, ∅-family 1 also contains a near optimal solution.

Once again we find that all of the (best) solutions within a given

| | |
|---|---|
| $k^o_a =$ | 113 |
| $k^o_p =$ | 113 |
| $k^o_r =$ | 22 |
| $m_{r_p} =$ | 4 |
| $k^o_1 =$ | 1 |
| $k^o_2 =$ | 1 |
| $k^o_3 =$ | 1 |
| $k^o_4 =$ | 4 |
| $k^o_7 =$ | 1 |
| $k^o_8 =$ | 3 |
| $k^o_9 =$ | 12 |
| $k^o_{10} =$ | 1 |

Table 6-14. Alternative Values for Parameters of Heart Disease Problem

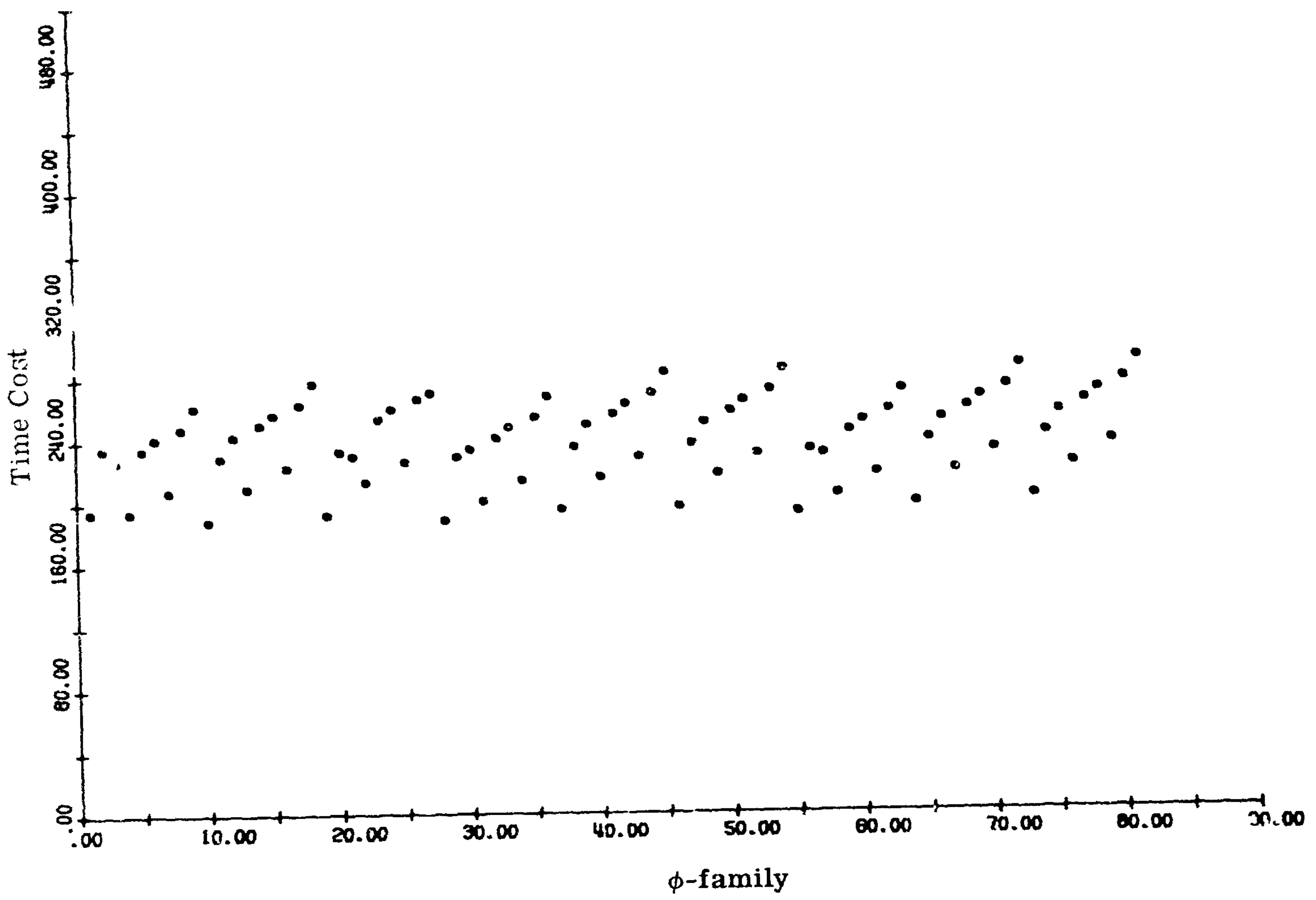


Figure 6-14. Time Costs by ϕ-family for Case 2

$\phi$-family: 28 (Optimal Solutions)

Solution 1

|  |  |
|---|---|
| $\phi_1 \cdots \phi_{10}$: | 0 0 1 0 0 0 0 0 0 0 |
| $\phi'_1 \cdots \phi'_{10}$: | 0 0 1 0 0 0 0 0 0 0 |
| $\Delta_1 \cdots \Delta_{10}$: | 1 0 0 1 1 0 1 1 1 1 |
| $\beta_2 \cdots \beta_{10}$: | 0 0 0 1 0 0 1 1 1 |
| $k_1 \cdots k_{10}$: | 1 4 1 1 1 3 1 1 1 1 |

Solution 2

|  |  |
|---|---|
| $\phi_1 \cdots \phi_{10}$: | 0 0 1 0 0 0 0 0 0 0 |
| $\phi'_1 \cdots \phi'_{10}$: | 0 0 1 0 0 0 0 0 0 0 |
| $\Delta_1 \cdots \Delta_{10}$: | 1 0 0 1 1 1 1 0 1 1 |
| $\beta_2 \cdots \beta_{10}$: | 0 0 0 1 0 1 0 0 1 |
| $k_1 \cdots k_{10}$: | 1 4 1 1 1 1 1 3 1 1 |

Common Items

$m_a = 1$ $m_p = 12$ $m_{r_1} = 1$ $m_{r_2} = 1$

$t_{3,1} = 84.00$

$t_{7,1} = 213.04$ $T = 188.39$

$t_{12,1} = 202.00$ $S = 22098$

$t_{14,1} = 344.00$

Table 6-15. Optimal and Near Optimal Solutions for Case 2

**$\phi$-family: 10** (Optimal Solutions)

**Solution 1**

| | |
|---|---|
| $\phi_1 \cdots \phi_{10}$: | 0 0 0 0 1 0 0 0 0 0 |
| $\phi'_1 \cdots \phi'_{10}$: | 0 0 0 0 1 0 0 0 0 0 |
| $\Delta_1 \cdots \Delta_{10}$: | 1 0 1 1 0 0 1 1 1 1 |
| $\beta_2 \cdots \beta_{10}$: | 0 0 1 0 0 0 1 1 1 |
| $k_1 \cdots k_{10}$: | 1 4 1 1 1 3 1 1 1 1 |

**Solution 2**

| | |
|---|---|
| $\phi_1 \cdots \phi_{10}$: | 0 0 0 0 1 0 0 0 0 0 |
| $\phi'_1 \cdots \phi'_{10}$: | 0 0 0 0 1 0 0 0 0 0 |
| $\Delta_1 \cdots \Delta_{10}$: | 1 0 1 1 0 1 1 0 1 1 |
| $\beta_2 \cdots \beta_{10}$: | 0 0 1 0 0 1 0 0 1 |
| $k_1 \cdots k_{10}$: | 1 4 1 1 1 1 1 3 1 1 |

**Solution 3**

| | |
|---|---|
| $\phi_1 \cdots \phi_{10}$: | 0 0 0 0 1 0 0 0 0 0 |
| $\phi'_1 \cdots \phi'_{10}$: | 0 0 0 0 1 0 0 0 0 0 |
| $\Delta_1 \cdots \Delta_{10}$: | 1 1 1 0 0 0 1 1 1 1 |
| $\beta_2 \cdots \beta_{10}$: | 1 1 0 0 0 0 1 1 1 |
| $k_1 \cdots k_{10}$: | 1 1 1 4 1 3 1 1 1 1 |

Table 6-15. Optimal and Near Optimal Solutions for Case 2 (Cont.)

Solution 4

|  |  |
|---|---|
| $\phi_1 \cdots \phi_{10}$: | 0 0 0 0 1 0 0 0 0 0 |
| $\phi'_1 \cdots \phi'_{10}$: | 0 0 0 0 1 0 0 0 0 0 |
| $\Delta_1 \cdots \Delta_{10}$: | 1 1 1 0 0 1 1 0 1 1 |
| $\beta_2 \cdots \beta_{10}$: | 1 1 0 0 0 1 0 0 1 |
| $k_1 \cdots k_{10}$: | 1 1 1 4 1 1 1 3 1 1 |

Common Items

$m_a = 1$ $m_p = 12$ $m_{r_1} = 1$ $m_{r_2} = 1$

$t_{3,1} = 84.00$

$t_{7,1} = 213.04$ $T = 188.39$

$t_{12,1} = 202.00$ $S = 22098$

$t_{14,1} = 344.00$

Table 6-15. Opti[illegible] [illegible] Near Optimal Solutions for Case [illegible] Cont.)

$\phi$-family: 19

Solution 1

| | |
|---|---|
| $\phi_1 \cdots \phi_{10}$: | 0 0 0 1 0 0 0 0 0 0 |
| $\phi'_1 \cdots \phi'_{10}$: | 0 0 0 1 0 0 0 0 0 0 |
| $\Delta_1 \cdots \Delta_{10}$: | 1 1 1 0 1 0 1 1 1 1 |
| $\beta_2 \cdots \beta_{10}$: | 1 1 0 0 0 0 1 1 1 |
| $k_1 \cdots k_{10}$: | 1 1 1 4 1 3 1 1 1 1 |

Solution 2

| | |
|---|---|
| $\phi_1 \cdots \phi_{10}$: | 0 0 0 1 0 0 0 0 0 0 |
| $\phi'_1 \cdots \phi'_{10}$: | 0 0 0 1 0 0 0 0 0 0 |
| $\Delta_1 \cdots \Delta_{10}$: | 1 1 1 0 1 1 1 0 1 1 |
| $\beta_2 \cdots \beta_{10}$: | 1 1 0 0 0 1 0 0 1 |
| $k_1 \cdots k_{10}$: | 1 1 1 4 1 1 1 3 1 1 |

Common Items

$m_a = 1$ $m_p = 12$ $m_{r_1} = 1$ $m_{r_2} = 1$

| | | | |
|---|---|---|---|
| $t_{3,1} =$ | 86.00 | | |
| $t_{7,1} =$ | 217.04 | $T =$ | 191.88 |
| $t_{12,1} =$ | 198.00 | $S =$ | 20742 |
| $t_{14,1} =$ | 236.00 | | |

Table 6-15. Optimal and Near Optimal Solutions for Case 2 (Cont.)

$\phi$-family: 55

Solution 1

| | |
|---|---|
| $\phi_1 \cdots \phi_{10}$: | 0 1 0 0 0 0 0 0 0 0 |
| $\phi'_1 \cdots \phi'_{10}$: | 0 1 0 0 0 0 0 0 0 0 |
| $\Delta_1 \cdots \Delta_{10}$: | 1 0 1 1 1 0 1 1 1 1 |
| $\beta_2 \cdots \beta_{10}$: | 0 0 1 1 0 0 1 1 1 |
| $k_1 \cdots k_{10}$: | 1 4 1 1 1 3 1 1 1 1 |

Solution 2

| | |
|---|---|
| $\phi_1 \cdots \phi_{10}$: | 0 1 0 0 0 0 0 0 0 0 |
| $\phi'_1 \cdots \phi'_{10}$: | 0 1 0 0 0 0 0 0 0 0 |
| $\Delta_1 \cdots \Delta_{10}$: | 1 0 1 1 1 1 1 0 1 1 |
| $\beta_2 \cdots \beta_{10}$: | 0 0 1 1 0 1 0 0 1 |
| $k_1 \cdots k_{10}$: | 1 4 1 1 1 1 1 3 1 1 |

Common Items

$m_a = 1$ $m_p = 12$ $m_{r_1} = 1$ $m_{r_2} = 1$

$t_{3,1} =$ 86.00

$t_{7,1} =$ 217.04 $T = 191.88$

$t_{12,1} =$ 198.00 $S = 20742$

$t_{14,1} =$ 236.00

Table 6-15. Optimal and Near Optimal Solutions for Case 2 (Cont.)

$\phi$-family: 4

Solution 1

| | |
|---|---|
| $\phi_1 \cdots \phi_{10}$: | 0 0 0 0 0 0 1 0 0 0 |
| $\phi'_1 \cdots \phi'_{10}$: | 0 0 0 0 0 0 1 0 0 0 |
| $\Delta_1 \cdots \Delta_{10}$: | 1 0 1 1 1 1 0 0 1 1 |
| $\beta_2 \cdots \beta_{10}$: | 0 0 1 1 0 0 0 0 1 |
| $k_1 \cdots k_{10}$: | 1 4 1 1 1 1 1 3 1 1 |

Solution 2

| | |
|---|---|
| $\phi_1 \cdots \phi_{10}$: | 0 0 0 0 0 0 1 0 0 0 |
| $\phi'_1 \cdots \phi'_{10}$: | 0 0 0 0 0 0 1 0 0 0 |
| $\Delta_1 \cdots \Delta_{10}$: | 1 1 1 0 1 1 0 0 1 1 |
| $\beta_2 \cdots \beta_{10}$: | 1 1 0 0 0 0 0 0 1 |
| $k_1 \cdots k_{10}$: | 1 1 1 4 1 1 1 3 1 1 |

Common Items

$m_a = 1$ $m_\rho = 12$ $m_{r_1} = 1$ $m_{r_2} = 1$

| | | | |
|---|---|---|---|
| $t_{3,1} =$ | 80.00 | | |
| $t_{7,1} =$ | 221.04 | T = | 194.07 |
| $t_{12,1} =$ | 202.00 | S = | 22098 |
| $t_{14,1} =$ | 344.00 | | |

Table 6-15. Optimal and Near Optimal Solutions for Case 2 (Cont.)

$\phi$-family: 37

Solution 1

| | |
|---|---|
| $\phi_1 \cdots \phi_{10}$: | 0 0 1 0 1 0 0 0 0 0 |
| $\phi'_1 \cdots \phi_{10}$: | 0 0 1 0 1 0 0 0 0 0 |
| $\Delta_1 \cdots \Delta_{10}$: | 1 1 0 0 0 0 1 1 1 1 |
| $\beta_2 \cdots \beta_{10}$: | 1 0 0 0 0 0 1 1 1 |
| $k_1 \cdots k_{10}$: | 1 1 1 4 1 3 1 1 1 1 |

Solution 2

| | |
|---|---|
| $\phi_1 \cdots \phi_{10}$: | 0 0 1 0 1 0 0 0 0 0 |
| $\phi'_1 \cdots \phi'_{10}$: | 0 0 1 0 1 0 0 0 0 0 |
| $\Delta_1 \cdots \Delta_{10}$: | 1 1 0 0 0 1 1 0 1 1 |
| $\beta_2 \cdots \beta_{10}$: | 1 0 0 0 0 1 0 0 1 |
| $k_1 \cdots k_{10}$: | 1 1 1 4 1 1 1 3 1 1 |

Common Items

$m_a = 1$ $m_p = 12$ $m_{r_1} = 1$ $m_{r_2} = 1$

$t_{3,1} =$ 90.00

$t_{7,1} =$ 219.04 $T = 194.50$

$t_{12,1} =$ 208.00 $S = 23454$

$t_{14,1} =$ 464.00

Table 6-15. Optimal and Near Optimal Solutions for Case 2 (Cont.)

$\phi$-family: 1

| | |
|---|---|
| $\phi_1 \cdots \phi_{10}$: | 0 0 0 0 0 0 0 0 0 0 |
| $\phi'_1 \cdots \phi'_{10}$: | 0 0 0 1 0 0 0 0 0 0 |
| $\Delta_1 \cdots \Delta_{10}$: | 1 1 1 0 1 0 1 1 1 1 |
| $\beta_2 \cdots \beta_{10}$: | 1 1 0 0 0 0 1 1 1 |
| $k_1 \cdots k_{10}$: | 1 1 1 4 1 3 1 1 1 1 |

$m_a = 1$ $m_p = 12$ $m_{r_1} = 1$ $m_{r_2} = 1$

| | | | |
|---|---|---|---|
| $t_{3,1} =$ | 83.00 | | |
| $t_{7,1} =$ | 221.04 | $T =$ | 194.51 |
| $t_{12,1} =$ | 178.00 | $S =$ | 18482 |
| $t_{14,1} =$ | 236.00 | | |

Table 6-15. Optimal and Near Optimal Solutions for Case 2 (Cont.)

$\phi$-family: 46

Solution 1

| | |
|---|---|
| $\phi_1 \cdots \phi_{10}$: | 0 0 1 1 0 0 0 0 0 0 |
| $\phi'_1 \cdots \phi'_{10}$: | 0 0 1 1 0 0 0 0 0 0 |
| $\Delta_1 \cdots \Delta_{10}$: | 1 1 0 0 1 0 1 1 1 1 |
| $\beta_2 \cdots \beta_{10}$: | 1 0 0 0 0 0 1 1 1 |
| $k_1 \cdots k_{10}$: | 1 1 1 4 1 3 1 1 1 1 |

Solution 2

| | |
|---|---|
| $\phi_1 \cdots \phi_{10}$: | 0 0 1 1 0 0 0 0 0 0 |
| $\phi'_1 \cdots \phi'_{10}$: | 0 0 1 1 0 0 0 0 0 0 |
| $\Delta_1 \cdots \Delta_{10}$: | 1 1 0 0 1 1 1 0 1 1 |
| $\beta_2 \cdots \beta_{10}$: | 1 0 0 0 0 1 0 0 1 |
| $k_1 \cdots k_{10}$: | 1 1 1 4 1 1 1 3 1 1 |

Common Items

$m_a = 1$ $m_p = 12$ $m_{r_1} = 1$ $m_{r_2} = 1$

$t_{3,1} = 90.00$

$t_{7,1} = 221.04$ $T = 196.00$

$t_{12,1} = 202.00$ $S = 22098$

$t_{14,1} = 356.00$

Table 6-15. Optimal and Near Optimal Solutions for Case 2 (Cont.)

ø-family result in the same physical storage structure. Moreover, the structures for ø-families 10 and 28 are again identical, as are the structures for ø-families 19 and 55. Figures 6-15 through 6-20 contain schematic representations for the solutions of ø-families 28 (and 10), 19 (and 55), 4, 37, 1, and 46, respectively.

Upon close inspection of the storage structures for ø-families 28, 19, 4, 37, and 46 we find that they are the same as the structures obtained for those ø-families in the previous case with the exception that the stacks of description block indicators do not contain head pointers. The lack of these head pointers is due, of course, to the reduced number of description block indicators in each stack.

We also see that the additional near optimal **solution** (in ø-family 1) has the same basic form as the others. Instead of having a ring (or stack of rings) or stacks of description block indicators, this structure uses a stack for the stacks of description block indicators.

For the purposes of comparison we have determined the time of the Case 1 optimal solution given the conditions of Case 2. The value of this time cost is 188.41, whereas the time cost of the Case 2 optimal solution (given the conditions of Case 2) is 188.39.

Thus, we may conclude that our solution is relatively insensitive to minor variations in the values assigned to the parameters $k_1^o \cdots k_{10}^o$.

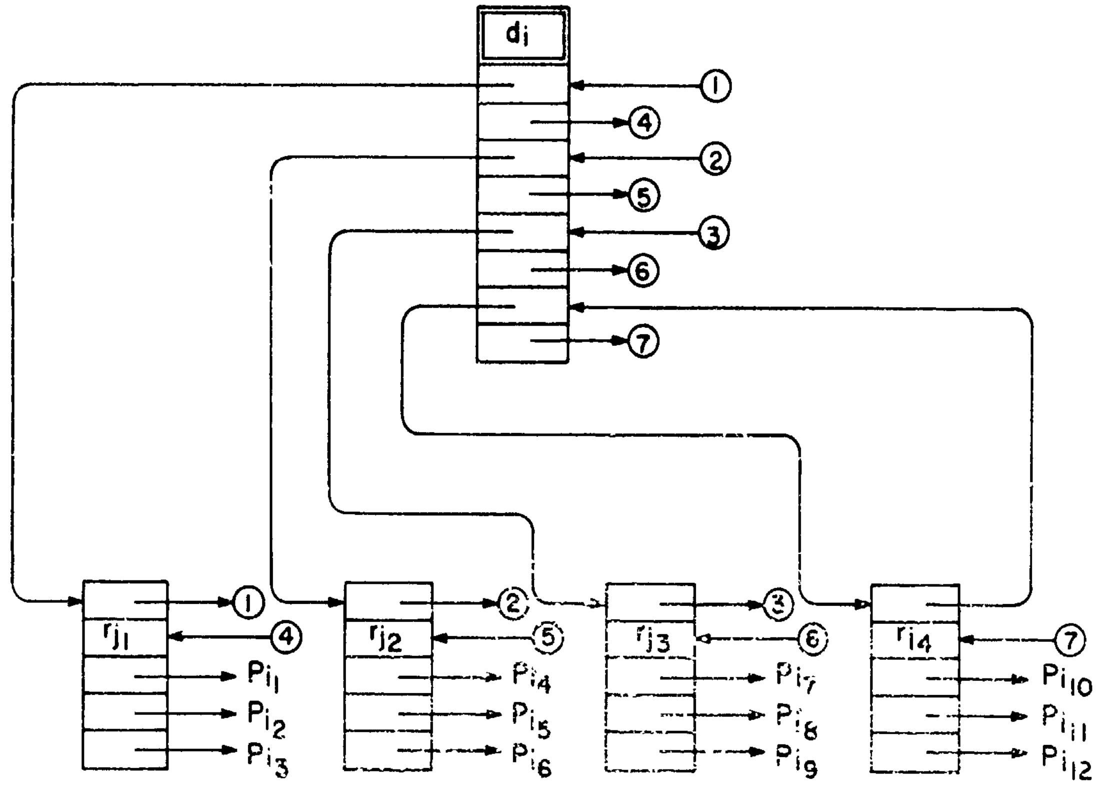


Figure 6-15. Best Solution for $\phi$-families 10 and 28 for Case 2 (Optimal Solution)

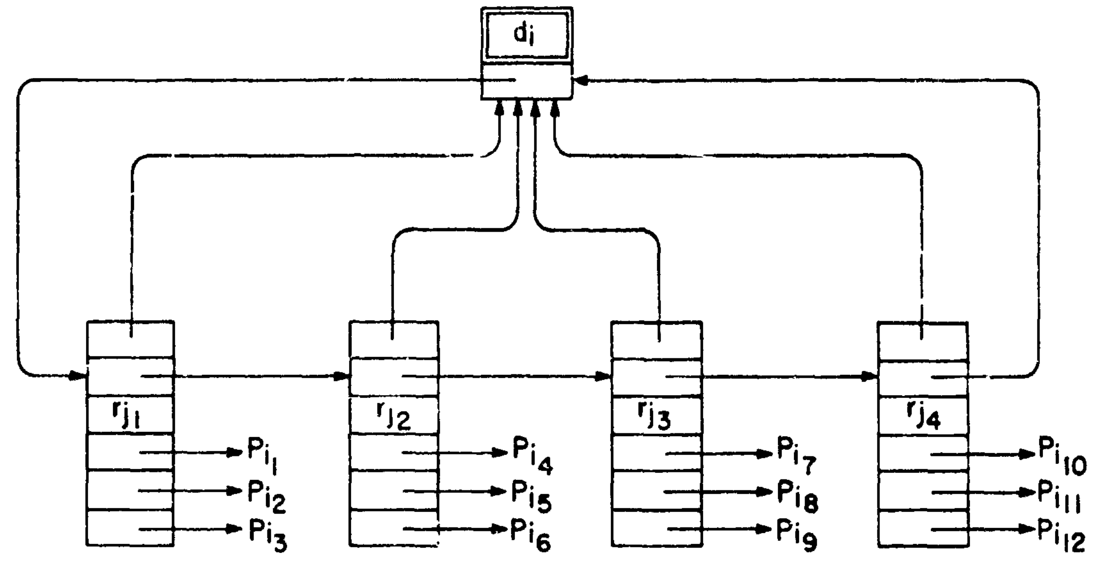


Figure 6-16. Best Solution for $\phi$-families 19 and 55 for Case 2

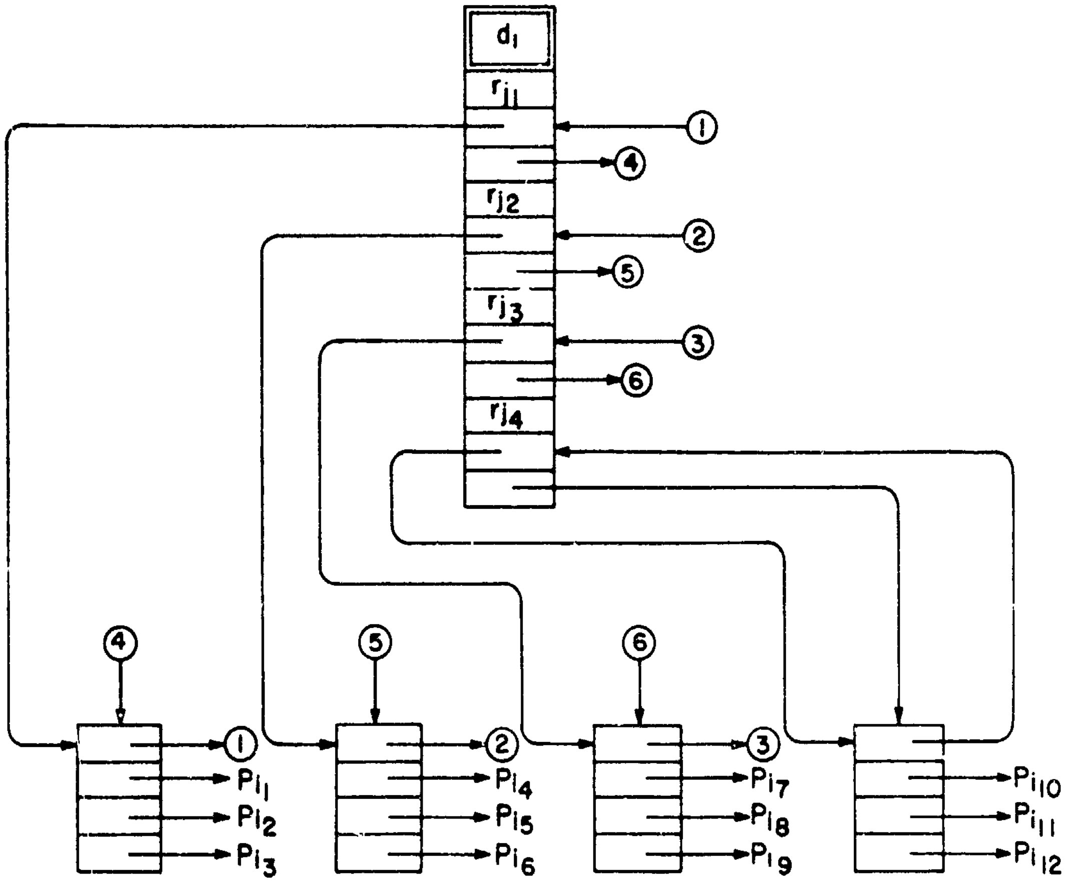


Figure 6-17. Best Solution for $\phi$-family 4 for Case 2

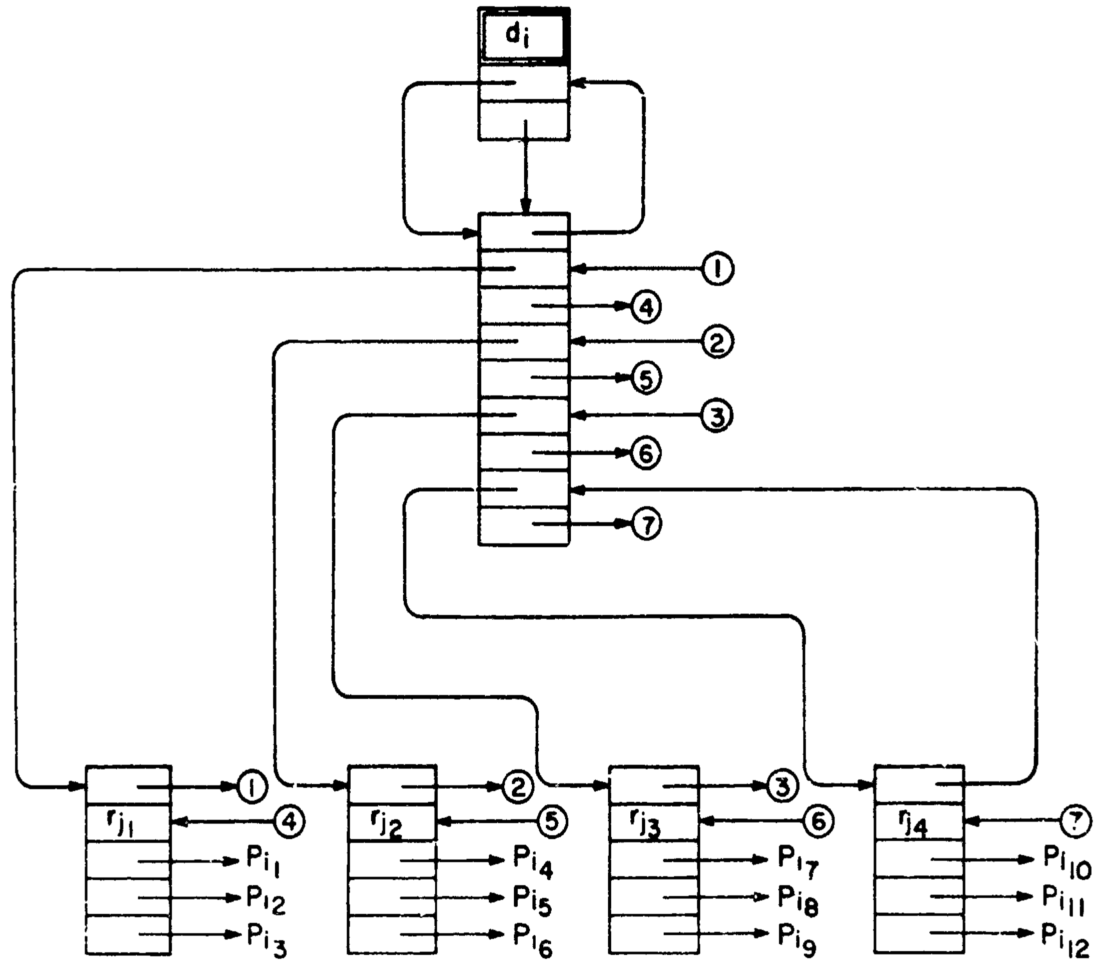


Figure 6-18. Best Solution for $\phi$-family 37 for Case 2

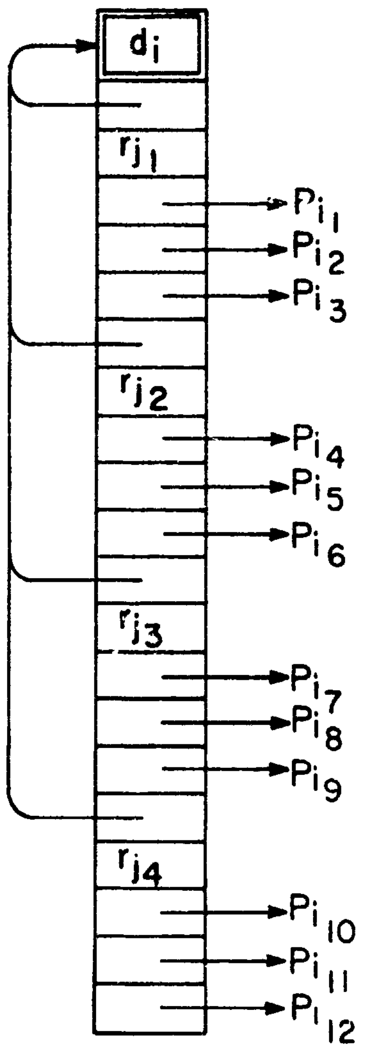


Figure 6-19. Best Solution for $\phi$-family 1 for Case 2

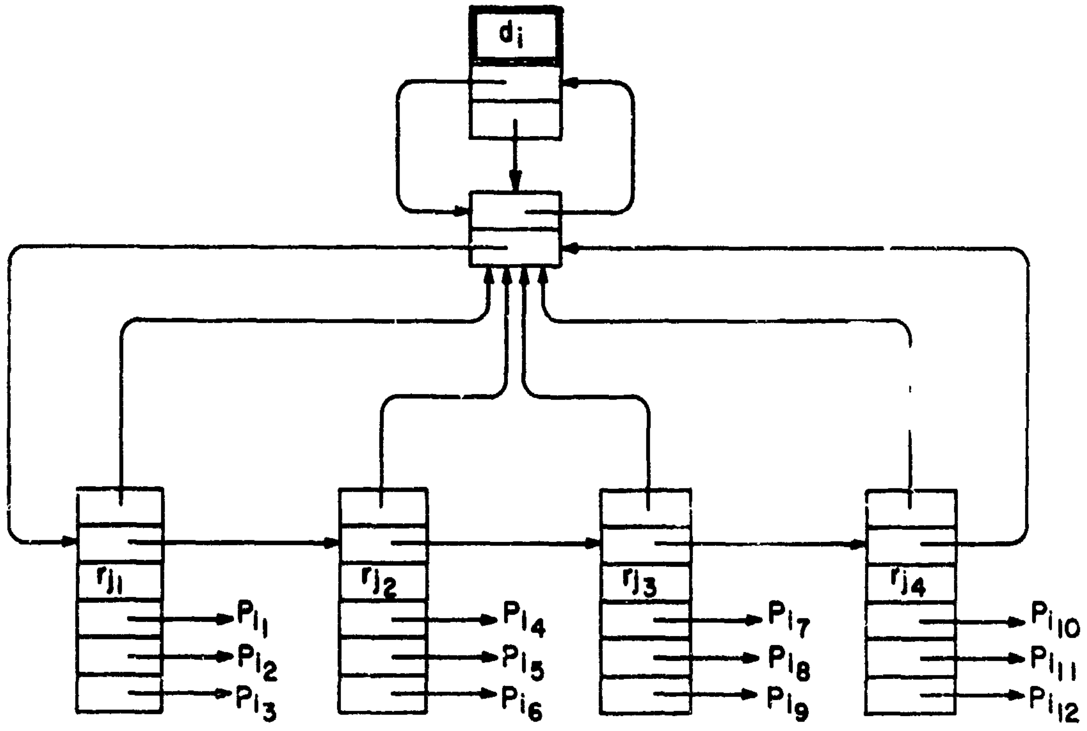


Figure 6-20. Best Solution for $\phi$-family 46 for Case 2

## 6.3.2 Variation in the Frequencies of Operations

Let us now consider the effects of variation in the relative frequencies of the given primitive operations. Instead of the relative frequencies used for Case 1, suppose that we were to use the following values:

| Operation | Relative Frequency |
|---|---|
| $Q_3(d_i r_j -)$ | 0.20 |
| $Q_7(d_i - p_k)$ | 0.70 |
| $Q_{12}(d_i * -)$ | 0.05 |
| $Q_{14}(- * p_k)$ | 0.05 |

These values tend to assign more weight then before to the less significant operations.

Keeping the values of all other parameters fixed as they were for the Case 1 and applying our program to these data, which we will designate Case 3, yields the solution listing contained in Appendix G. Figure 6-21 contains a plot of the time costs for the best solutions in each of the various ϕ-families.

It appears for this case that the optimal solution lies in a somewhat sharper valley since there are fewer solutions within 3 to 4 percent of the optimum (although this fact may not be overly significant). We see once again that the optimal and near optimal solutions are

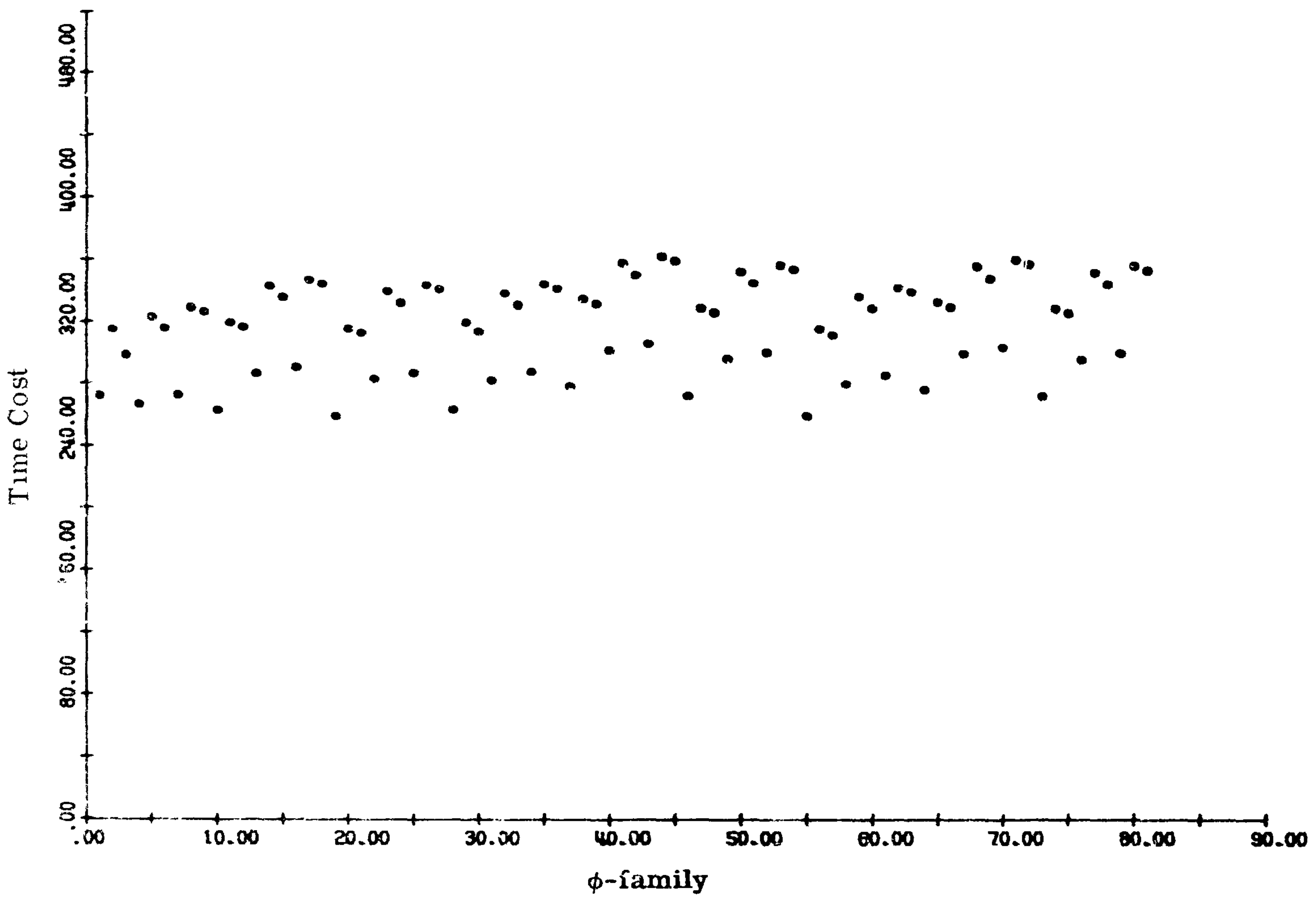


Figure 6-21. Time Costs by $\phi$-family for Case 3

concentrated in the same $\phi$-families, as before, however. In particular the $\phi$-families containing solutions within 3 percent of the optimum are 4, 10, 19, 28, and 55.

We see also that these solutions are identical to those in the corresponding $\phi$-families for Case 1. The only difference is in the ordering of these solutions (by time cost). For this case $\phi$-families 19 and 55 contain the optimal solution, followed by the solutions in $\phi$-families 10 and 28 and in $\phi$-family 4, in that order.

We see that given the conditions of Case 3 the Case 1 optimal solution has a time cost of 262.95 versus a time cost of 258.85 for the Case 3 optimal solution.

We conclude, therefore, that our solution is also relatively insensitive to minor variations in the relative frequencies assigned to the given primitive operations, although it is perhaps more sensitive to variations in the values of these parameters than to variations in the values of $k_1^0 \cdots k_{10}^0$.

## 6.3.3 Solution for the Primary Operation

We may find it instructive to compare the solutions we have obtained using a weighted sum of the time costs for several operations and average statistics for relations with a fairly large variance with the solutions we might obtain for a single primitive operation and statistics for only the relations with which that operation is intimately concerned.

In particular, let us consider the primitive operation $Q_7$ (which has the greatest relative frequency in our other examples) and the relations $r^o_1 \cdots r^o_{20}$. We can easily determine that the relations $r^o_1 \cdots r^o_{20}$ will result in the statistics of Table 6-16. (See the derivation of the results of Table 6-7 for the procedure involved.) Table 6-17 contains a suitable integer approximation of these statistics.

Appendix H contains the solution listing for this problem and Figure 6-22 contains a plot of the time costs for the best solutions in the various ø-families.

The optimal storage structure, which is given by the solutions for ø-families 10 and 28, appears in Figure 6-23. The ø-families which contain solutions with time costs within 3 percent of the optimum are, in order of increasing time cost, 19, 55, 1, 37, and 46.

From this information it is clear that in our earlier examples operation $Q_7$ has, because of its weight, a very marked effect upon deciding what the optimal structure is.

## 6.4 Summary

Let us conclude this example by summarizing its main points. First, we have shown that application of our procedure to a particular problem can result in the specification of a storage structure significantly more efficient than a storage structure determined by other less rigorous, qualitative methods.

| | | |
|---|---|---|
| $k_a^o$ | = | 50 |
| $k_p^o$ | = | 33 |
| $k_r^o$ | = | 20 |
| | | |
| $m_{r_p}$ | = | 2.86 |
| | | |
| $k_1^o$ | = | 1 |
| $k_2^o$ | = | 1 |
| $k_3^o$ | = | 1 |
| $k_4^o$ | = | 6.64 |
| | | |
| $k_7^o$ | = | 1 |
| $k_8^o$ | = | 2.82 |
| $k_9^o$ | = | 28.39 |
| $k_{10}^o$ | = | 1 |

Table 6-16. Characteristics of Relations $r_1^o \cdots r_{20}^o$

| | | |
|---|---|---|
| $k_a^o$ | = | 50 |
| $k_p^o$ | = | 30 |
| $k_r^o$ | = | 20 |
| $m_{r_p}$ | = | 3 |
| $k_1^o$ | = | 1 |
| $k_2^o$ | = | 1 |
| $k_3^o$ | = | 1 |
| $k_4^o$ | = | 6 |
| $k_7^o$ | = | 1 |
| $k_8^o$ | = | 3 |
| $k_9^o$ | = | 30 |
| $k_{10}^o$ | = | 1 |

Table 6-17. Integer Approximation of Statistics in Table 6-16.

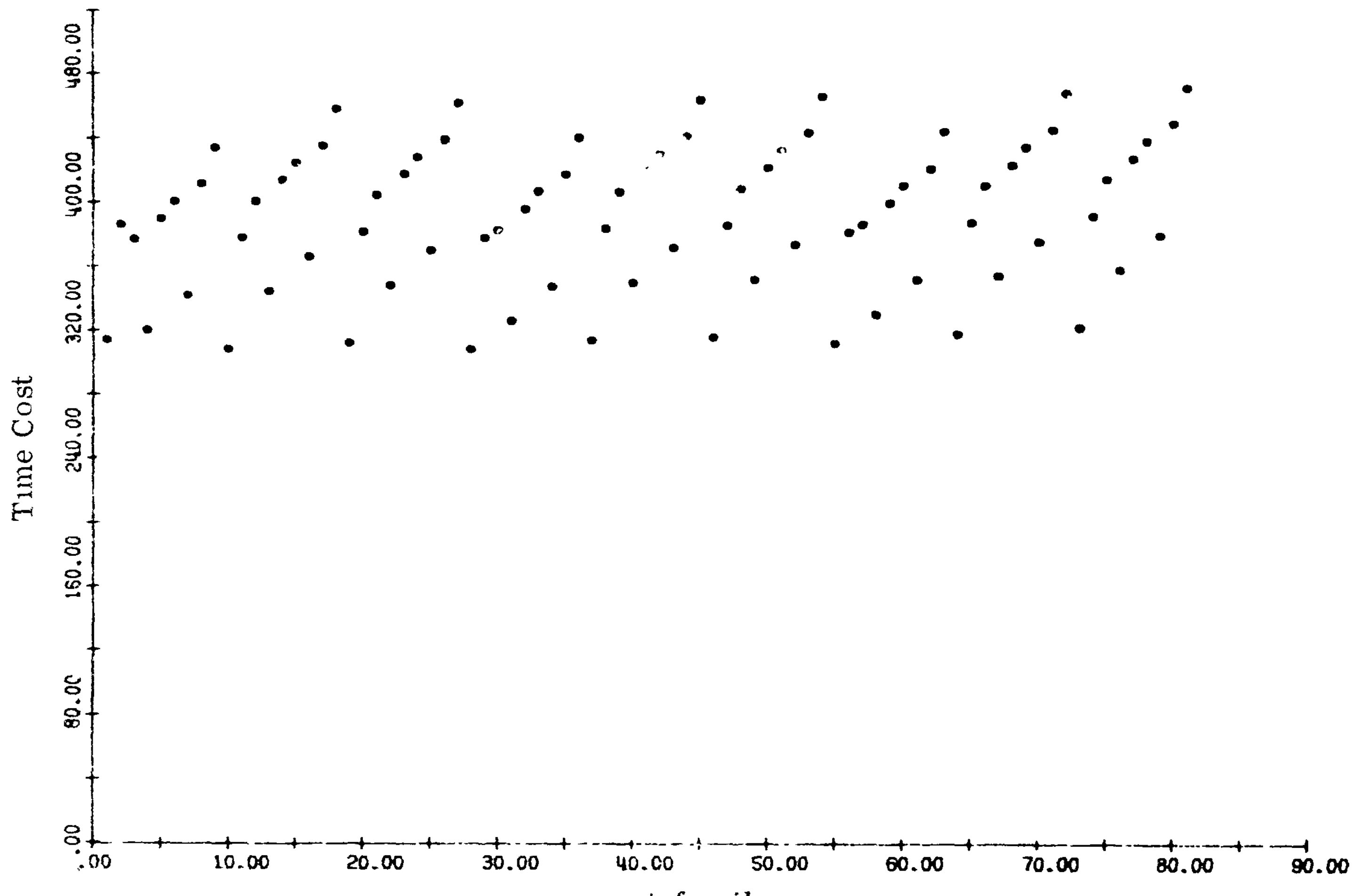


Figure 6-22. Time Costs by $\phi$-family for Operation $Q_7$

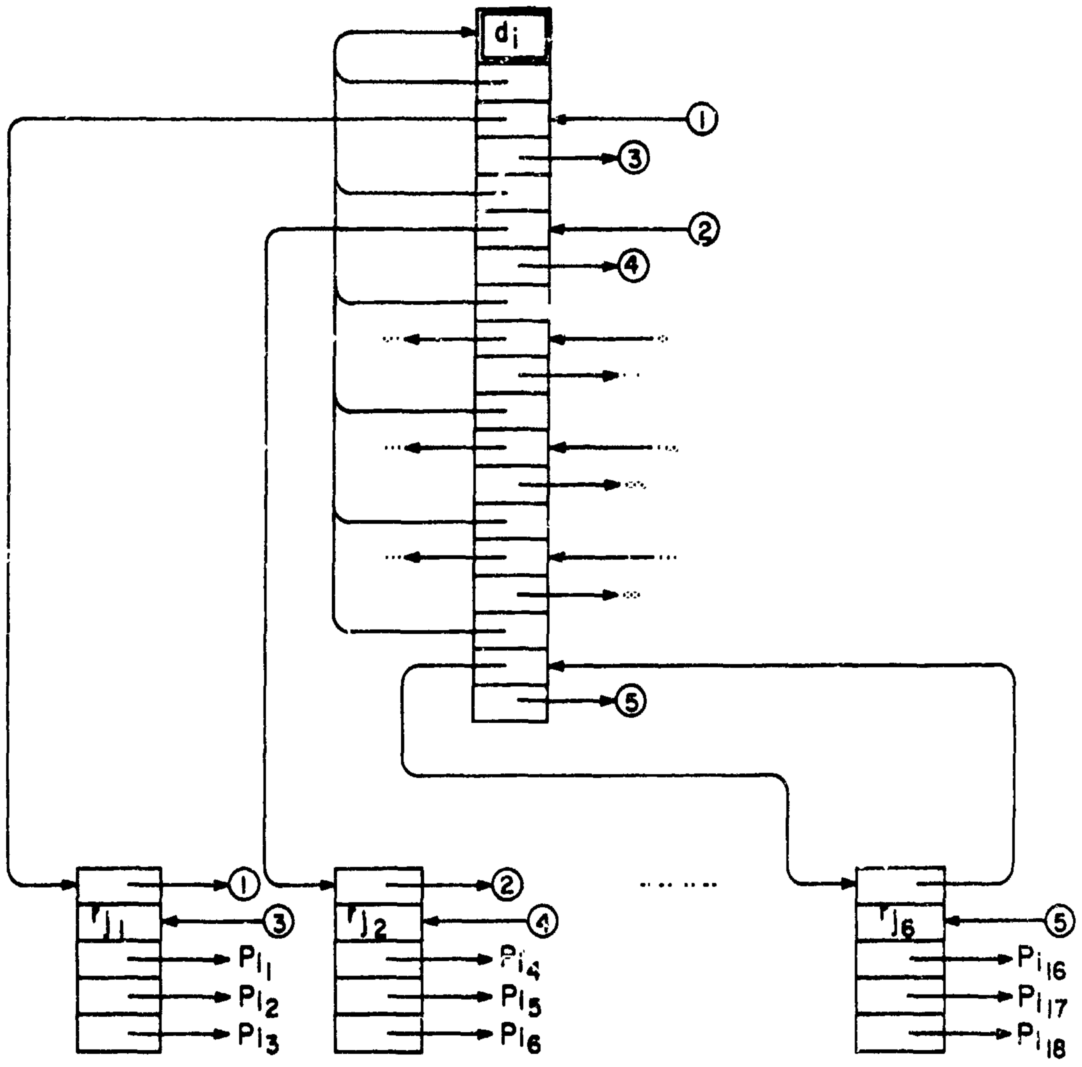


Figure 6-23. Optimal Solution for Primitive Operation $Q_7$

Secondly, we have illustrated that the solution obtained is relatively insensitive to minor variations in the values of the parameters $k_1^0 \cdots k_{10}^0$ and the relative frequencies of the given primitive operations. This is of particular significance since the estimation of these values may be somewhat difficult for problems involving large numbers of data items and relations.

Finally, at least within the context of the problem considered, we have demonstrated that the optimal storage structure is very much dependent upon the operation having the greatest weight. This information may be of importance if all the operations to be used in conjunction with the structure are not known with complete certainty - the primary operation (or operations) may then be used to obtain a solution which can be at least near optimal in terms of the overall problem.

# Chapter VII

# CONCLUSION

The objective of the research reported here has been the development of a rigorous quantitative method for the automatic design of optimal storage structures for the representation of data within a computer memory.

To accomplish this objective we partitioned the problem into four basic parts. First, in order to provide a framework within which to work, we defined a relational model of data structure. To apply this model to a particular problem, the user (i.e., the problem solver) must determine what data items are involved in the solution process he has chosen and what relations (of accessibility) relate the various data items to one another. He must then determine the (average) cardinalities of the various sets which we have defined for the model. These cardinalities in essence give us a measure of the redundancy or the "share-ability" of the relations among the data.

Second, we developed a model for the specification of the storage structures which can represent an arbitrary data structure as given by the data structure model. This storage structure model consists primarily of a set of decision variables, which are used to specify the structural form of a storage structure, and a set of parameters, which are used to characterize the environment (i.e., the computer) in which the storage structure is to exist.

As a third step, we defined two basic measures of performance - a time cost function and a storage cost function - for use in comparing the relative merits of a collection of storage structures. The time cost function reflects the number of time units required to perform certain primitive operations using a particular storage structure, where the problem solver chooses the primitive operations of interest from a standard collection of these primitive operations and assigns weights to them to reflect their relative importance in the problem solution. The storage cost function simply reflects the total number of storage units occupied by the storage structure of interest.

Finally, we presented a procedure which (automatically) compares the time and storage costs for those storage structures which are members of the set of feasible storage structures (for a given data structure) and determines the storage structure for which these costs satisfy certain optimality conditions. For our purposes, a storage structure is considered to be optimal if it minimizes the time cost function, subject to the constraint that its storage cost is less than some bound.

In order to demonstrate the feasibility and the effectiveness of the techniques which we developed, we applied them to a problem for which a solution program had already been implemented and for which fairly extensive data about system performance were available. The results we obtained were enlightening and demonstrate conclusively that the approach we have taken is not only feasible, but desirable.

First, we were able to show that the storage structure which we determined to be optimal reduces by a factor of approximately 2.5 the time required to solve a typical problem, as compared with the storage structure implemented for the existing system. The optimal storage structure also requires approximately two-thirds the storage required by the existing storage structure, even though we made no particular attempt to minimize storage and assumed in fact no limit on the storage available. Although such significant improvement in system performance may not always result, these figures do give some indication of the potential of a rigorous approach to storage structure design.

Next we considered the "uniqueness" of the optimal storage structure. The results we obtained indicate that there are a number of storage structures with time costs within 3 or 4 percent of the time cost for the optimal solution (and with essentially the same storage costs). This would tend to indicate that there is no single storage structure which should be considered <u>the</u> optimum, but rather that there are a number of near-optimal storage structures, any one of which would serve as well as the others. To a certain extent, this is true. We found on the other hand, however, that all of these near-optimal storage structures were very nearly identical to the storage structure determined to be the optimum. This fact tends to favor the idea of a single storage structure.

We also considered the sensitivity of our solution to variations in the values of those parameters most subject to error - namely, the average values of the cardinalities of the data structure sets and the relative frequencies of the primitive operations. The values of these parameters must be estimated by the problem solver and, hence, are subject to the errors inherent in any estimation process. This will be particularly true for the cardinalities of the sets when the data structure involves very large numbers of data items and relations. Our results indicate, however, that the solution we obtain is relatively insensitive to minor variations in either of these two sets of parameters.

Finally, we examined the solution obtained by considering only the primitive operation with the greatest relative frequency (0.8 out of 1.0) and compared this result with the previously determined solution. On the basis of this comparison we concluded that when all the primitive operations for a particular problem are not known with a great degree of certainty, we can obtain a solution which may be very nearly optimal by considering only the principal primitive operation or operations (which are presumably known).

To further investigate the sensitivity of the results of our storage structure design procedure, it might be desirable to consider a more extensive variety of problems, perhaps a broader range of variation in the values of the parameters to which the sensitivity calculations apply, and possibly variations in the values of additional parameters (such as the relative times to follow a pointer and to step through a stack). We cannot argue with the potential value of such investigations. We can only point out that each variation in the problem to be solved requires performing the complete solution process, which for our prototype program involves, while not excessive, a nontrivial amount of computer time. In particular, solution of our example problem involving four primitive operations required approximately 7.3 minutes of CPU time on the IBM 360/67. This suggests that one should at least choose his examples and the values of his parameters for these investigations in a prudent manner.

While on the subject of the amount of computer time required by the solution process, we might point out that the amount of time required is completely independent of the numbers of data items and relations (and the corresponding cardinalities of the data structure sets) and is a function primarily of the number of primitive operations characterizing the problem to be solved. This follows directly, of course, from the nature of our time cost function.

Let us now consider some of the strengths and weaknesses of certain design considerations employed in developing our procedure.

In order to apply the procedure developed here to his particular problem, the user must specify the data structure which characterizes his problem. This involves specifying the numbers of data items and relations involved in his problem, as well as the average cardinalities of the various data structure sets.

Before he can even begin to determine these quantities, however, the user must rigorously define the solution process which he intends to use for his problem and in doing so must choose the relations which will characterize his data structure. By assuming that we must be given the data structure specification as the starting point of our storage structure design procedure, we have completely ignored this facet of the user's system design problem. Specification of a solution process is, of course, another of the areas of computer systems design for which there is a paucity of formal, objective techniques for evaluation and comparison of the effects of alternative decisions. Since the storage structures specified by our procedure are strongly influenced by the user's choice of relations (i.e., by his choice of solution process), this matter is clearly of importance to us in spite of our assumed disregard for it.

Let us discuss for a moment the implications of using average values for the cardinalities of the various data structure sets. In the first place, using exact values for these cardinalities is completely out of the question in all but the simplest of situations. Determining exact values would be tantamount to constructing a schematic of the entire data structure for a given problem, and applying our procedure to a data structure model utilizing exact values would essentially be equivalent to completely solving the user's problem for every storage structure considered. Clearly, even for problems of moderate size, neither of these steps can be accomplished through a reasonable amount of effort.

On the other hand, determining average values for the cardinalities can be done quite easily (as we have shown in our example). Furthermore, using average values tends to minimize the amount of information which must be supplied by the user.

One might criticize the use of average values if the variances of the various cardinalities are large, but it may be possible in such cases to partition the problem (i.e., the data structure) into sub-problems for which the variances are not so great, and then to determine the optimal storage structure for each of these sub-problems.

Closely allied with the specification of the relations and the set cardinalities for the data structure is the selection of primitive operations and their relative frequencies. Since we have provided only a limited number of primitive operations, the user may encounter some difficulty in trying to match exactly his operations with the primitive operations. The obvious solution to this problem is, of course, to define additional primitive operations. This may or may not be desirable, depending upon how great the mismatch is.

Perhaps the biggest weakness in our consideration of the storage structure problem is the neglect of manipulative operations. Most certainly the updating and alteration of the information contained in a storage structure can have a profound effect upon the form of that storage structure. Again, the obvious solution to this problem is simply to define additional primitive operations to cover the operations desired. This can certainly be done, but development of the corresponding time costs would definitely require a nontrivial amount of effort.

We hasten to point out that there are a multitude of problems which involve only the interrogation of a static data base. Clearly, our procedure can be applied to these problems without the necessity of adding manipulative operations to our set of primitive operations.

One other matter with which one might take issue is our choice of optimality conditions. Although other choices may be made, we feel that minimization of the time cost function subject to a limit on the storage available is the most universally applicable.

Finally, we have assumed that the storage structures considered by our design procedure are to reside in the primary store of the computer. By introducing appropriate constraints and by choosing proper values for the parameters representing time cost characteristics, it should also be possible to apply the model developed here to problems for which the storage structures are to reside in the secondary store or a hierarchy of storage devices.

## Appendix A

## METHODS OF IMPLEMENTATION FOR THE PRIMITIVE OPERATIONS

Operation $Q_1$: $d_i\ r_j\ p_k$

Method 1

Search for source $d_i$.

Search for $r_j$ among the associated relation symbols.

If $r_j$ is found, search for $p_k$ among the associated targets.

Method 2

Search for relation $r_j$.

Search for $d_i$ among the associated sources.

If $d_i$ is found, search for $p_k$ among the associated targets.

Method 3

Search for relation $r_j$.

Search for $p_k$ among the associated targets.

If $p_k$ is found, search for $d_i$ among the associated sources.

Method 4

Search for target $p_k$.

Search for $r_j$ among the associated relation symbols.

If $r_j$ is found, search for $d_i$ among the associated sources.

Operation $Q_2$: $d_i \; r_j \; *$

Method 1

Search for source $d_i$.

Search for $r_j$ among the associated relation symbols.

Method 2

Search for relation $r_j$.

Search for $d_i$ among the associated sources.

Operation $Q_3$: $d_i \; r_j$ -

Method 1

Search for source $d_i$.

Search for $r_j$ among the associated relation symbols.

Determine all associated targets.

Method 2

Search for relation $r_j$.

Search for $d_i$ among the associated sources.

Determine all associated targets.

Method 3

Determine all targets.

For each target, search for $r_j$ among the associated relation symbols.

If $r_j$ is found, search for $d_i$ among the associated sources.

Operation $Q_4$: $* \ r_j \ p_k$

Method 1

Search for relation $r_j$.

Search for $p_k$ among the associated targets.

Method 2

Search for target $p_k$.

Search for $r_j$ among the associated relation symbols.

Operation $Q_5$: $- \ r_j \ p_k$

Method 1

Search for relation $r_j$.

Search for $p_k$ among the associated targets.

Determine all associated sources.

Method 2

Search for target $p_k$.

Search for $r_j$ among the associated relation symbols.

Determine all associated sources.

Method 3

Determine all sources.

For each source, search for $r_j$ among the associated relation symbols.

If $r_j$ is found, search for $p_k$ among the associated targets.

Operation $Q_6$: $d_i * p_k$

Method 1

Search for source $d_i$.

Search for $p_k$ among the associated targets.

Method 2

Search for target $p_k$.

Search for $d_i$ among the associated sovrces.

Operation $Q_7$: $d_i - p_k$

Method 1

Search for source $d_i$.

Determine all associated relation symbols.

For each relation symbol, search for $p_k$ among the associated targets.

Method 2

Search for target $p_k$.

Determine all associated relation symbols.

For each relation symbol, search for $d_i$ among the associated sources.

Method 3

Determine all relations.

For each relation, search for source $d_i$.

If $d_i$ is found, search for $p_k$ among the associated targets.

Method 4

Determine all relations.

For each relation, search for target $p_k$.

If $p_k$ is found, search for $d_i$ among the associated source-

Operation $Q_8$: $- \ r_j \ *$

Method 1

Search for relation $r_j$.

Determine all associated sources.

Method 2

Determine all sources.

For each source, search for $r_j$ among the associated relation symbols.

Operation $Q_9$: $* \ r_j \ -$

Method 1

Search for relation $r_j$.

Determine all associated targets.

Method 2

Determine all targets.

For each target, search for $r_j$ among the associated relation symbols.

Operation $Q_{10}$: $- \ r_j \ -$

Method 1

Search for relation $r_j$.

Determine all associated sources.

Determine all associated targets.

Method 2

Search for relation $r_j$.

Determine all associated targets.

Determine all associated sources.

Method 3

Determine all sources.

For each source, search for $r_j$ among the associated relation symbols.

If $r_j$ is found, determine all associated targets.

Method 4

Determine all targets.

For each target, search for $r_j$ among the associated relation symbols.

If $r_j$ is found, determine all associated sources.

Operation $Q_{11}$: $d_i$ - *

Method 1

Search for source $d_i$.

Determine all associated relation symbols.

Method 2

Determine all relations.

For each relation, search for $d_i$ among the associated sources.

Operation $Q_{12}$: $d_i$ * -

Method 1

Search for source $d_i$.

Determine all associated targets.

Method 2

Determine all targets.

For each target, search for $d_i$ among the associated sources.

Operation $Q_{13}$: $d_i$ - - †

Method 1

Search for source $d_i$.

Determine all associated relation symbols.

Determine all associated targets.

Method 2

Determine all relations.

For each relation, search for $d_i$ among the associated sources.

If $d_i$ is found, determine all associated targets.

Operation $Q_{14}$: - * $p_k$

Method 1

Search for target $p_k$.

Determine all associated sources.

Method 2

Determine all sources.

For each source, search for $p_k$ among the associated targets.

Operation $Q_{15}$: $* - p_k$

Method 1

Search for target $p_k$.

Determine all associated relation symbols.

Method 2

Determine all relations.

For each relation, search for $p_k$ among the associated targets.

Operation $Q_{16}$: $- - p_k$ †

Method 1

Search for target $p_k$.

Determine all associated relation symbols.

Determine all associated sources.

Method 2

Determine all relations.

For each relation, search for $p_k$ among the associated targets.

If $p_k$ is found, determine all associated sources.

†Two additional methods could be defined for each of these operations ($Q_{13}$ and $Q_{16}$) but were judged always to be more costly than the methods given here (since each involves searching the entire storage structure) and are, therefore, not considered.

## Appendix B

## TABULATION OF $e_i$ FOR $i \in \{1, 2, \cdots, 10\}$

Each of the functions $e_i$ for $i \in \{1, 2, \cdots, 10\}$ is presented here in tabular form as a function of $\phi_1, \phi_2, \ldots, \phi_{10}, \Delta_1, \Delta_2, \ldots, \Delta_{10}$.

The variables for which values are unspecified in a given row of a table are free to assume either the value 1 or the value 0 (subject, of course, to the constraints of Chapter III). Note, however, that values specified for $\phi_1, \phi_2, \ldots, \phi_{10}$ pertain to all succeeding rows of a table (even though not explicitly specified) until new values are assigned to them.

For all (legal) combinations of values not covered by its corresponding table a function has the value 0.

| $\phi_1$ | $\phi_2$ | $\phi_3$ | $\phi_4$ | $\phi_5$ | $\phi_6$ | $\phi_7$ | $\phi_8$ | $\phi_9$ | $\phi_{10}$ | $\Delta_1$ | $\Delta_2$ | $\Delta_3$ | $\Delta_4$ | $\Delta_5$ | $\Delta_6$ | $\Delta_7$ | $\Delta_8$ | $\Delta_9$ | $\Delta_{10}$ | $e_2$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 1 | | | | | | | | | | | | | | | | | | | $f_2$ |
| | 0 | 1 | | | | | | | | | 0 | | | | | | | | | $s_2$ |
| | 0 | 0 | 1 | | | | | | | | 0 | 1 | | | | | | | | $s_2$ |
| | 0 | 0 | 0 | 1 | | | | | | | 0 | 1 | 1 | | | | | | | $s_2$ |
| | | | | | | | | | | | 0 | 1 | 0 | | | | | | | $S_1(2,4)$ |
| | 0 | 0 | 0 | 0 | 1 | | | | | | 0 | 1 | 1 | 1 | | | | | | $s_2$ |
| | | | | | | | | | | | 0 | 1 | 0 | 1 | | | | | | $S_1(2,4)$ |
| | 0 | 0 | 0 | 0 | 0 | 1 | | | | | 0 | 1 | 1 | 1 | 1 | | | | | $s_2$ |
| | | | | | | | | | | | 0 | 1 | 1 | 1 | 0 | | | | | $S_1(2,6)$ |
| | | | | | | | | | | | 0 | 1 | 0 | 1 | 1 | | | | | $S_1(2,4)$ |
| | | | | | | | | | | | 0 | 1 | 0 | 1 | 0 | | | | | $S_2(2,4,6)$ |
| | 0 | 0 | 0 | 0 | 0 | 0 | 1 | | | | 0 | 1 | 1 | 1 | 1 | 1 | | | | $s_2$ |
| | | | | | | | | | | | 0 | 1 | 1 | 1 | 0 | 1 | | | | $S_1(2,6)$ |
| | | | | | | | | | | | 0 | 1 | 0 | 1 | 1 | 1 | | | | $S_1(2,4)$ |
| | | | | | | | | | | | 0 | 1 | 0 | 1 | 0 | 1 | | | | $S_2(2,4,6)$ |

Table B-1. Elementary Time Cost $e_2$

| $\phi_1 \phi_2 \phi_3 \phi_4 \phi_5 \phi_6 \phi_7 \phi_8 \phi_9 \phi_{10}$ | $\Delta_1 \Delta_2 \Delta_3 \Delta_4 \Delta_5 \Delta_6 \Delta_7 \Delta_8 \Delta_9 \Delta_{10}$ | $e_2$ |
|---|---|---|
| 0 0 0 0 0 0 0 1 | 0 1 1 1 1 1 1 | $s_2$ |
| | 0 1 1 1 1 1 0 | $S_1(2,8)$ |
| | 0 1 1 1 0 1 1 | $S_1(2,6)$ |
| | 0 1 1 1 0 1 0 | $S_2(2,6,8)$ |
| | 0 1 0 1 1 1 1 | $S_1(2,4)$ |
| | 0 1 0 1 1 1 0 | $S_2(2,4,8)$ |
| | 0 1 0 1 0 1 1 | $S_2(2,4,6)$ |
| | 0 1 0 1 0 1 0 | $S_3(2,4,6,8)$ |
| 0 0 0 0 0 0 0 0 1 | 0 1 1 1 1 1 1 1 | $s_2$ |
| | 0 1 1 1 1 1 0 1 | $S_1(2,8)$ |
| | 0 1 1 1 0 1 1 1 | $S_1(2,6)$ |
| | 0 1 1 1 0 1 0 1 | $S_2(2,6,8)$ |
| | 0 1 0 1 1 1 1 1 | $S_1(2,4)$ |
| | 0 1 0 1 1 1 0 1 | $S_2(2,4,8)$ |
| | 0 1 0 1 0 1 1 1 | $S_2(2,4,6)$ |
| | 0 1 0 1 0 1 0 1 | $S_3(2,4,6,8)$ |

Table B-1. Elementary Time Cost $e_2$ (Cont.)

| $\phi_1$ | $\phi_2$ | $\phi_3$ | $\phi_4$ | $\phi_5$ | $\phi_6$ | $\phi_7$ | $\phi_8$ | $\phi_9$ | $\phi_{10}$ | $\Delta_1$ | $\Delta_2$ | $\Delta_3$ | $\Delta_4$ | $\Delta_5$ | $\Delta_6$ | $\Delta_7$ | $\Delta_8$ | $\Delta_9$ | $\Delta_{10}$ | $e_2$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
|  | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |  | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | $s_2$ |
|  |  |  |  |  |  |  |  |  |  |  | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | $S_1(2,10)$ |
|  |  |  |  |  |  |  |  |  |  |  | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | $S_1(2,8)$ |
|  |  |  |  |  |  |  |  |  |  |  | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | $S_2(2,8,10)$ |
|  |  |  |  |  |  |  |  |  |  |  | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | $S_1(2,6)$ |
|  |  |  |  |  |  |  |  |  |  |  | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | $S_2(2,6,10)$ |
|  |  |  |  |  |  |  |  |  |  |  | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | $S_2(2,6,8)$ |
|  |  |  |  |  |  |  |  |  |  |  | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | $S_3(2,6,8,10)$ |
|  |  |  |  |  |  |  |  |  |  |  | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | $S_1(2,4)$ |
|  |  |  |  |  |  |  |  |  |  |  | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | $S_2(2,4,10)$ |
|  |  |  |  |  |  |  |  |  |  |  | 0 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | $S_2(2,4,8)$ |
|  |  |  |  |  |  |  |  |  |  |  | 0 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 0 | $S_3(2,4,8,10)$ |
|  |  |  |  |  |  |  |  |  |  |  | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | $S_2(2,4,6)$ |
|  |  |  |  |  |  |  |  |  |  |  | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 0 | $S_3(2,4,6,10)$ |
|  |  |  |  |  |  |  |  |  |  |  | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | $S_3(2,4,6,8)$ |
|  |  |  |  |  |  |  |  |  |  |  | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | $S_4(2,4,6,8,10)$ |

Table B-1 Elementary Time Cost $e_2$ (Cont.)

| $\phi_1$ | $\phi_2$ | $\phi_3$ | $\phi_4$ | $\phi_5$ | $\phi_6$ | $\phi_7$ | $\phi_8$ | $\phi_9$ | $\phi_{10}$ | $\Delta_1$ | $\Delta_2$ | $\Delta_3$ | $\Delta_4$ | $\Delta_5$ | $\Delta_6$ | $\Delta_7$ | $\Delta_8$ | $\Delta_9$ | $\Delta_{10}$ | $e_4$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
|  |  |  | 1 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | $f_4$ |
|  |  |  | 0 | 1 |  |  |  |  |  |  |  |  | 0 |  |  |  |  |  |  | $s_4$ |
|  |  |  | 0 | 0 | 1 |  |  |  |  |  |  |  | 0 | 1 |  |  |  |  |  | $s_4$ |
|  |  |  | 0 | 0 | 0 | 1 |  |  |  |  |  |  | 0 | 1 | 1 |  |  |  |  | $s_4$ |
|  |  |  |  |  |  |  |  |  |  |  |  |  | 0 | 1 | 0 |  |  |  |  | $S_1(4,6)$ |
|  |  |  | 0 | 0 | 0 | 0 | 1 |  |  |  |  |  | 0 | 1 | 1 | 1 |  |  |  | $s_4$ |
|  |  |  |  |  |  |  |  |  |  |  |  |  | 0 | 1 | 0 | 1 |  |  |  | $S_1(4,6)$ |
|  |  |  | 0 | 0 | 0 | 0 | 0 | 1 |  |  |  |  | 0 | 1 | 1 | 1 | 1 |  |  | $s_4$ |
|  |  |  |  |  |  |  |  |  |  |  |  |  | 0 | 1 | 1 | 1 | 0 |  |  | $S_1(4,8)$ |
|  |  |  |  |  |  |  |  |  |  |  |  |  | 0 | 1 | 0 | 1 | 1 |  |  | $S_1(4,6)$ |
|  |  |  |  |  |  |  |  |  |  |  |  |  | 0 | 1 | 0 | 1 | 0 |  |  | $S_2(4,6,8)$ |
|  |  |  | 0 | 0 | 0 | 0 | 0 | 0 | 1 |  |  |  | 0 | 1 | 1 | 1 | 1 | 1 |  | $s_4$ |
|  |  |  |  |  |  |  |  |  |  |  |  |  | 0 | 1 | 1 | 1 | 0 | 1 |  | $S_1(4,8)$ |
|  |  |  |  |  |  |  |  |  |  |  |  |  | 0 | 1 | 0 | 1 | 1 | 1 |  | $S_1(4,6)$ |
|  |  |  |  |  |  |  |  |  |  |  |  |  | 0 | 1 | 0 | 1 | 0 | 1 |  | $S_2(4,6,8)$ |
|  |  |  | 0 | 0 | 0 | 0 | 0 | 0 | 0 |  |  |  | 0 | 1 | 1 | 1 | 1 | 1 | 1 | $s_4$ |
|  |  |  |  |  |  |  |  |  |  |  |  |  | 0 | 1 | 1 | 1 | 1 | 1 | 0 | $S_1(4,10)$ |
|  |  |  |  |  |  |  |  |  |  |  |  |  | 0 | 1 | 1 | 1 | 0 | 1 | 1 | $S_1(4;8)$ |
|  |  |  |  |  |  |  |  |  |  |  |  |  | 0 | 1 | 1 | 1 | 0 | 1 | 0 | $S_2(4,8,10)$ |
|  |  |  |  |  |  |  |  |  |  |  |  |  | 0 | 1 | 0 | 1 | 1 | 1 | 1 | $S_1(4,6)$ |
|  |  |  |  |  |  |  |  |  |  |  |  |  | 0 | 1 | 0 | 1 | 1 | 1 | 0 | $S_2(4,6,,10)$ |
|  |  |  |  |  |  |  |  |  |  |  |  |  | 0 | 1 | 0 | 1 | 0 | 1 | 1 | $S_2(4,6,8)$ |
|  |  |  |  |  |  |  |  |  |  |  |  |  | 0 | 1 | 0 | 1 | 0 | 1 | 0 | $S_3(4,6,8,10)$ |

Table B-2. Elementary Time Cost $e_4$

| $\phi_1$ | $\phi_2$ | $\phi_3$ | $\phi_4$ | $\phi_5$ | $\phi_6$ | $\phi_7$ | $\phi_8$ | $\phi_9$ | $\phi_{10}$ | $\Delta_1$ | $\Delta_2$ | $\Delta_3$ | $\Delta_4$ | $\Delta_5$ | $\Delta_6$ | $\Delta_7$ | $\Delta_8$ | $\Delta_9$ | $\Delta_{10}$ | $e_6$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | 1 | | | | | | | | | | | | | | | $f_6$ |
| | | | | | 0 | 1 | | | | | | | | | 0 | | | | | $s_6$ |
| | | | | | 0 | 0 | 1 | | | | | | | | 0 | 1 | | | | $s_3$ |
| | | | | | 0 | 0 | 0 | 1 | | | | | | | 0 | 1 | 1 | | | $s_6$ |
| | | | | | | | | | | | | | | | 0 | 1 | 0 | | | $S_1(6,8)$ |
| | | | | | 0 | 0 | 0 | 0 | 1 | | | | | | 0 | 1 | 1 | 1 | | $s_6$ |
| | | | | | | | | | | | | | | | 0 | 1 | 0 | 1 | | $S_1(6,8)$ |
| | | | | | 0 | 0 | 0 | 0 | 0 | | | | | | 0 | 1 | 1 | 1 | 1 | $s_6$ |
| | | | | | | | | | | | | | | | 0 | 1 | 1 | 1 | 0 | $S_1(6,10)$ |
| | | | | | | | | | | | | | | | 0 | 1 | 0 | 1 | 1 | $S_1(6,8)$ |
| | | | | | | | | | | | | | | | 0 | 1 | 0 | 1 | 0 | $S_2(6,8,10)$ |

Table B-3. Elementary Time Cost $e_6$

| $\phi_1$ | $\phi_2$ | $\phi_3$ | $\phi_4$ | $\phi_5$ | $\phi_6$ | $\phi_7$ | $\phi_8$ | $\phi_9$ | $\phi_{10}$ | $\Delta_1$ | $\Delta_2$ | $\Delta_3$ | $\Delta_4$ | $\Delta_5$ | $\Delta_6$ | $\Delta_7$ | $\Delta_8$ | $\Delta_9$ | $\Delta_{10}$ | $e_8$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | 1 | | | | | | | | | | | | | $f_8$ |
| | | | | | | | 0 | 1 | | | | | | | | | 0 | | | $s_8$ |
| | | | | | | | 0 | 0 | 1 | | | | | | | | 0 | 1 | | $s_8$ |
| | | | | | | | 0 | 0 | 0 | | | | | | | | 0 | 1 | 1 | $s_8$ |
| | | | | | | | | | | | | | | | | | 0 | 1 | 0 | $S_1(8,10)$ |

Table B-4. Elementary Time Cost $e_8$

| | $\phi_1$ | $\phi_2$ | $\phi_3$ | $\phi_4$ | $\phi_5$ | $\phi_6$ | $\phi_7$ | $\phi_8$ | $\phi_9$ | $\phi_{10}$ | $\Delta_1$ | $\Delta_2$ | $\Delta_3$ | $\Delta_4$ | $\Delta_5$ | $\Delta_6$ | $\Delta_7$ | $\Delta_8$ | $\Delta_9$ | $\Delta_{10}$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $e_{10}$ | | | | | | | | | | | | | | | | | | | | |
| $f_{10}$ | | | | | | | | | | 1 | | | | | | | | | | |
| $s_{10}$ | | | | | | | | | | 0 | | | | | | | | | | 0 |

Table B-5. **Elementary Time Cost $e_{10}$**

| $\phi_1$ | $\phi_2$ | $\phi_3$ | $\phi_4$ | $\phi_5$ | $\phi_6$ | $\phi_7$ | $\phi_8$ | $\phi_9$ | $\phi_{10}$ | $\Delta_1$ | $\Delta_2$ | $\Delta_3$ | $\Delta_4$ | $\Delta_5$ | $\Delta_6$ | $\Delta_7$ | $\Delta_8$ | $\Delta_9$ | $\Delta_{10}$ | $e_9$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | 1 | | | | | | | | | | | | $f_9$ |
| | | | | | | | 1 | 0 | | | | | | | | | | 0 | | $s_9$ |
| | | | | | | 1 | 0 | 0 | | | | | | | | | 1 | 0 | | $s_9$ |
| | | | | | 1 | 0 | 0 | 0 | | | | | | | | 1 | 1 | 0 | | $s_9$ |
| | | | | | | | | | | | | | | | | 0 | 1 | 0 | | $S_1(9, 7)$ |
| | | | | 1 | 0 | 0 | 0 | 0 | | | | | | | 1 | 1 | 1 | 0 | | $s_9$ |
| | | | | | | | | | | | | | | | 1 | 0 | 1 | 0 | | $S_1(9, 7)$ |
| | | | 1 | 0 | 0 | 0 | 0 | 0 | | | | | | 1 | 1 | 1 | 1 | 0 | | $s_9$ |
| | | | | | | | | | | | | | | 0 | 1 | 1 | 1 | 0 | | $S_1(9, 5)$ |
| | | | | | | | | | | | | | | 1 | 1 | 0 | 1 | 0 | | $S_1(9, 7)$ |
| | | | | | | | | | | | | | | 0 | 1 | 0 | 1 | 0 | | $S_2(9, 7, 5)$ |
| | | 1 | 0 | 0 | 0 | 0 | 0 | 0 | | | | | 1 | 1 | 1 | 1 | 1 | 0 | | $s_9$ |
| | | | | | | | | | | | | | 1 | 0 | 1 | 1 | 1 | 0 | | $S_1(9, 5)$ |
| | | | | | | | | | | | | | 1 | 1 | 1 | 0 | 1 | 0 | | $S_1(9, 7)$ |
| | | | | | | | | | | | | | 1 | 0 | 1 | 0 | 1 | 0 | | $S_2(9, 7, 5)$ |

Table B-6. Elementary Time Cost $e_9$

| $\phi_1$ | $\phi_2$ | $\phi_3$ | $\phi_4$ | $\phi_5$ | $\phi_6$ | $\phi_7$ | $\phi_8$ | $\phi_9$ | $\phi_{10}$ | $\Delta_1$ | $\Delta_2$ | $\Delta_3$ | $\Delta_4$ | $\Delta_5$ | $\Delta_6$ | $\Delta_7$ | $\Delta_8$ | $\Delta_9$ | $\Delta_{10}$ | $e_9$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | | | | 1 | 1 | 1 | 1 | 1 | 1 | 0 | $s_9$ |
| | | | | | | | | | | | | | 0 | 1 | 1 | 1 | 1 | 1 | 0 | $S_1(9,3)$ |
| | | | | | | | | | | | | | 1 | 1 | 0 | 1 | 1 | 1 | 0 | $S_1(9,5)$ |
| | | | | | | | | | | | | | 0 | 1 | 0 | 1 | 1 | 1 | 0 | $S_2(9,5,3)$ |
| | | | | | | | | | | | | | 1 | 1 | 1 | 1 | 0 | 1 | 0 | $S_1(9,7)$ |
| | | | | | | | | | | | | | 0 | 1 | 1 | 1 | 0 | 1 | 0 | $S_2(9,7,3)$ |
| | | | | | | | | | | | | | 1 | 1 | 0 | 1 | 0 | 1 | 0 | $S_2(9,7,5)$ |
| | | | | | | | | | | | | | 0 | 1 | 0 | 1 | 0 | 1 | 0 | $S_3(9,7,5,3)$ |
| | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | | | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | $s_9$ |
| | | | | | | | | | | | | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | $S_1(9,3)$ |
| | | | | | | | | | | | | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | $S_1(9,5)$ |
| | | | | | | | | | | | | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 0 | $S_2(9,5,3)$ |
| | | | | | | | | | | | | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | $S_1(9,7)$ |
| | | | | | | | | | | | | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 0 | $S_2(9,7,3)$ |
| | | | | | | | | | | | | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | $S_2(9,7,5)$ |
| | | | | | | | | | | | | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | $S_3(9,7,5,3)$ |

Table B-6. Elementary Time Cost $e_9$ (Cont.)

| $\phi_1 \phi_2 \phi_3 \phi_4 \phi_5 \phi_6 \phi_7 \phi_8 \phi_9 \phi_{10}$ | $\Delta_1 \Delta_2 \Delta_3 \Delta_4 \Delta_5 \Delta_6 \Delta_7 \Delta_8 \Delta_9 \Delta_{10}$ | $e_9$ |
|---|---|---|
| 0 0 0 0 0 0 0 0 0 | 1 1 1 1 1 1 1 1 0 | $s_9$ |
| | 0 1 1 1 1 1 1 1 0 | $S_1(9,1)$ |
| | 1 1 0 1 1 1 1 1 0 | $S_1(9,3)$ |
| | 0 1 0 1 1 1 1 1 0 | $S_2(9,3,1)$ |
| | 1 1 1 1 0 1 1 1 0 | $S_1(9,5)$ |
| | 0 1 1 1 0 1 1 1 0 | $S_2(9,5,1)$ |
| | 1 1 0 1 0 1 1 1 0 | $S_2(9,5,3)$ |
| | 0 1 0 1 0 1 1 1 0 | $S_3(9,5,3,1)$ |
| | 1 1 1 1 1 1 0 1 0 | $S_1(9,7)$ |
| | 0 1 1 1 1 1 0 1 0 | $S_2(9,7,1)$ |
| | 1 1 0 1 1 1 0 1 0 | $S_2(9,7,3)$ |
| | 0 1 0 1 1 1 0 1 0 | $S_3(9,7,3,1)$ |
| | 1 1 1 1 0 1 0 1 0 | $S_2(9,7,5)$ |
| | 0 1 1 1 0 1 0 1 0 | $S_3(9,7,5,1)$ |
| | 1 1 0 1 0 1 0 1 0 | $S_3(9,7,5,3)$ |
| | 0 1 0 1 0 1 0 1 0 | $S_4(9,7,5,3,1)$ |

Table B-6. Elementary Time Cost $e_9$ (Cont.)

| $\phi_1$ | $\phi_2$ | $\phi_3$ | $\phi_4$ | $\phi_5$ | $\phi_6$ | $\phi_7$ | $\phi_8$ | $\phi_9$ | $\phi_{10}$ | $\Delta_1$ | $\Delta_2$ | $\Delta_3$ | $\Delta_4$ | $\Delta_5$ | $\Delta_6$ | $\Delta_7$ | $\Delta_8$ | $\Delta_9$ | $\Delta_{10}$ | $e_7$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | 1 | | | | | | | | | | | | | | $t_7$ |
| | | | | | 1 | 0 | | | | | | | | | | 0 | | | | $s_7$ |
| | | | | 1 | 0 | 0 | | | | | | | | | 1 | 0 | | | | $s_7$ |
| | | | 1 | 0 | 0 | 0 | | | | | | | | 1 | 1 | 0 | | | | $s_7$ |
| | | | | | | | | | | | | | | 0 | 1 | 0 | | | | $S_1(7,5)$ |
| | | 1 | 0 | 0 | 0 | 0 | | | | | | | 1 | 1 | 1 | 0 | | | | $s_7$ |
| | | | | | | | | | | | | | 1 | 0 | 1 | 0 | | | | $S_1(7,5)$ |
| | 1 | 0 | 0 | 0 | 0 | 0 | | | | | | 1 | 1 | 1 | 1 | 0 | | | | $s_7$ |
| | | | | | | | | | | | | 0 | 1 | 1 | 1 | 0 | | | | $S_1(7,3)$ |
| | | | | | | | | | | | | 1 | 1 | 0 | 1 | 0 | | | | $S_1(7,5)$ |
| | | | | | | | | | | | | 0 | 1 | 0 | 1 | 0 | | | | $S_2(7,5,3)$ |

Table B-7. Elementary Time Cost $e_7$

| $\phi_1$ | $\phi_2$ | $\phi_3$ | $\phi_4$ | $\phi_5$ | $\phi_6$ | $\phi_7$ | $\phi_8$ | $\phi_9$ | $\phi_{10}$ | $\Delta_1$ | $\Delta_2$ | $\Delta_3$ | $\Delta_4$ | $\Delta_5$ | $\Delta_6$ | $\Delta_7$ | $\Delta_8$ | $\Delta_9$ | $\Delta_{10}$ | $e_7$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | | | | | 1 | 1 | 1 | 1 | 1 | 0 | | | | $s_7$ |
| | | | | | | | | | | | 1 | 0 | 1 | 1 | 1 | 0 | | | | $S_1(7,3)$ |
| | | | | | | | | | | | 1 | 1 | 1 | 0 | 1 | 0 | | | | $S_1(7,5)$ |
| | | | | | | | | | | | 1 | 0 | 1 | 0 | 1 | 0 | | | | $S_2(7,5,3)$ |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | | | | 1 | 1 | 1 | 1 | 1 | 1 | 0 | | | | $s_7$ |
| | | | | | | | | | | 0 | 1 | 1 | 1 | 1 | 1 | 0 | | | | $S_1(7,1)$ |
| | | | | | | | | | | 1 | 1 | 0 | 1 | 1 | 1 | 0 | | | | $S_1(7,3)$ |
| | | | | | | | | | | 0 | 1 | 0 | 1 | 1 | 1 | 0 | | | | $S_2(7,3,1)$ |
| | | | | | | | | | | 1 | 1 | 1 | 1 | 0 | 1 | 0 | | | | $S_1(7,5)$ |
| | | | | | | | | | | 0 | 1 | 1 | 1 | 0 | 1 | 0 | | | | $S_2(7,5,1)$ |
| | | | | | | | | | | 1 | 1 | 0 | 1 | 0 | 1 | 0 | | | | $S_2(7,5,3)$ |
| | | | | | | | | | | 0 | 1 | 0 | 1 | 0 | 1 | 0 | | | | $S_3(7,5,3,1)$ |

Table B-7. Elementary Time Cost $e_7$ (Cont.)

| $\phi_1$ | $\phi_2$ | $\phi_3$ | $\phi_4$ | $\phi_5$ | $\phi_6$ | $\phi_7$ | $\phi_8$ | $\phi_9$ | $\phi_{10}$ | $\Delta_1$ | $\Delta_2$ | $\Delta_3$ | $\Delta_4$ | $\Delta_5$ | $\Delta_6$ | $\Delta_7$ | $\Delta_8$ | $\Delta_9$ | $\Delta_{10}$ | $e_5$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | 1 | | | | | | | | | | | | | | | | $f_5$ |
| | | | 1 | 0 | | | | | | | | | | 0 | | | | | | $s_5$ |
| | | 1 | 0 | 0 | | | | | | | | | 1 | 0 | | | | | | $s_5$ |
| | 1 | 0 | 0 | 0 | | | | | | | | 1 | 1 | 0 | | | | | | $s_5$ |
| | | | | | | | | | | | | 0 | 1 | 0 | | | | | | $S_1(5,3)$ |
| 1 | 0 | 0 | 0 | 0 | | | | | | | 1 | 1 | 1 | 0 | | | | | | $s_5$ |
| | | | | | | | | | | | 1 | 0 | 1 | 0 | | | | | | $S_1(5,3)$ |
| 0 | 0 | 0 | 0 | 0 | | | | | | 1 | 1 | 1 | 1 | 0 | | | | | | $s_5$ |
| | | | | | | | | | | 0 | 1 | 1 | 1 | 0 | | | | | | $S_1(5,1)$ |
| | | | | | | | | | | 1 | 1 | 0 | 1 | 0 | | | | | | $S_1(5,3)$ |
| | | | | | | | | | | 0 | 1 | 0 | 1 | 0 | | | | | | $S_2(5,3,1)$ |

Table E 8. Elementary Time Cost $e_5$

| $\phi_1$ | $\phi_2$ | $\phi_3$ | $\phi_4$ | $\phi_5$ | $\phi_6$ | $\phi_7$ | $\phi_8$ | $\phi_9$ | $\phi_{10}$ | $\Delta_1$ | $\Delta_2$ | $\Delta_3$ | $\Delta_4$ | $\Delta_5$ | $\Delta_6$ | $\Delta_7$ | $\Delta_8$ | $\Delta_9$ | $\Delta_{10}$ | $e_3$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | 1 | | | | | | | | | | | | | | | | | | $f_3$ |
| | 1 | 0 | | | | | | | | | | 0 | | | | | | | | $s_3$ |
| 1 | 0 | 0 | | | | | | | | | 1 | 0 | | | | | | | | $s_3$ |
| 0 | 0 | 0 | | | | | | | | 1 | 1 | 0 | | | | | | | | $s_3$ |
| | | | | | | | | | | 0 | 1 | 0 | | | | | | | | $S_1(3,1)$ |

Table B-9. Elementary Time Cost $e_3$

| $\phi_1$ | $\phi_2$ | $\phi_3$ | $\phi_4$ | $\phi_5$ | $\phi_6$ | $\phi_7$ | $\phi_8$ | $\phi_9$ | $\phi_{10}$ | $\Delta_1$ | $\Delta_2$ | $\Delta_3$ | $\Delta_4$ | $\Delta_5$ | $\Delta_6$ | $\Delta_7$ | $\Delta_8$ | $\Delta_9$ | $\Delta_{10}$ | $e_1$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | | | | | | | | | | | | | | | | | | | | $f_1$ |
| 0 | | | | | | | | | | 0 | | | | | | | | | | $s_1$ |

Table B-10. Elementary Time Cost $e_1$

## Appendix C

SUMMARY OF $z_i(t)$ and $z_i^*(t)$ FOR $i \in \{1, 2, \cdots, 10\}$

$$z_1(t) = e_6^* + e_5^o + K_{33}\,[\quad e_5' + \lambda_4(e_4^* + e_3^o)]$$

$$K_{32}\,[\lambda_4 e_3' + \lambda_2\,(e_2^* + e_1^o)]$$

$$K_{31}\,[\lambda_2 e_1' + t] + z_1^o$$

where

$$z_1^o = \{((\phi_6 \vee \phi_6') \vee \Delta_6\phi_5 \vee \Delta_6\Delta_5(\phi_4 \vee \phi_4') \vee \cdots \vee \Delta_6\Delta_5\Delta_4\Delta_3\Delta_2\Delta_1)$$

$$+ (\phi_6\overline{\phi}_6'\phi_5 \vee \phi_6\overline{\phi}_6'\Delta_5(\phi_4 \vee \phi_4') \vee \phi_6\overline{\phi}_6'\Delta_5\Delta_4\phi_3 \vee \cdots \vee \phi_6\overline{\phi}_6'\Delta_5\Delta_4\Delta_3\Delta_2\Delta_1)$$

$$+K_{33}[(\phi_5(\phi_4 \vee \phi_4') \vee \phi_5\Delta_4\phi_3 \vee \phi_5\Delta_4\Delta_3(\phi_2 \vee \phi_2') \vee \cdots \vee \phi_5\Delta_4\Delta_3\Delta_2\Delta_1)$$

$$+(\phi_4\overline{\phi}_4'\phi_3 \vee \phi_4\overline{\phi}_4'\Delta_3(\phi_2 \vee \phi_2') \vee \phi_4\overline{\phi}_4'\Delta_3\Delta_2\phi_1 \vee \phi_4\overline{\phi}_4'\Delta_3\Delta_2\Delta_1)]$$

$$+K_{32}[(\phi_3(\phi_2 \vee \phi_2') \vee \phi_3\Delta_2\phi_1 \vee \phi_3\Delta_2\Delta_1)$$

$$+ (\phi_2\overline{\phi}_2'\phi_1 \vee \phi_2\overline{\phi}_2'\Delta_1)]$$

$$+K_{31}\,[\phi_1]\}\,s_o$$

$$z_2(t) = e_5^o + K_{33}[\quad e_5' + \lambda_4 (e_4^* + e_3^o)]$$

$$+ K_{32}[\lambda_4 e_3' + \lambda_2(e_2^* + e_1^o)]$$

$$+ K_{31}[\lambda_2 e_1' + t] + z_2^o$$

where

$$z_2^o = \{(\phi_5 \vee \Delta_5(\phi_4 \vee \phi_4') \vee \Delta_5\Delta_4\phi_3 \vee \Delta_5\Delta_4\Delta_3(\phi_2 \vee \phi_2') \vee \cdots \vee \Delta_5\Delta_4\Delta_3\Delta_2\Delta_1)$$

$$+ K_{33}[(\phi_5(\phi_4 \vee \phi_4') \vee \phi_5\Delta_4\phi_3 \vee \phi_5\Delta_4\Delta_3(\phi_2 \vee \phi_2') \vee \cdots \vee \phi_5\Delta_4\Delta_3\Delta_2\Delta_1)$$

$$+(\phi_4\overline{\phi}_4'\phi_3 \vee \phi_4\overline{\phi}_4'\Delta_3(\phi_2 \vee \phi_2') \vee \phi_4\overline{\phi}_4'\Delta_3\Delta_2\phi_1 \vee \phi_4\overline{\phi}_4'\Delta_3\Delta_2\Delta_1)]$$

$$+ K_{32}[(\phi_3(\phi_2 \vee \phi_2') \vee \phi_3\Delta_2\phi_1 \vee \phi_3\Delta_2\Delta_1)$$

$$+(\phi_2\overline{\phi}_2'\phi_1 \vee \phi_2\overline{\phi}_2'\Delta_1)]$$

$$+ K_{31}[\phi_1] \quad \} s_o$$

$$z_3(t) = e_5^* + e_6^c + K_{34}\,[\;\; e_6' + \lambda_7(e_7^* + e_8^o)]$$

$$+ K_{35}\,[\,\lambda_7 e_8' + \lambda_9(e_9^* + e_{10}^o)]$$

$$+ K_{36}\,[\,\lambda_9 e_{10}' + t\,] \;+ z_3^o$$

where

$$z_3^o = \{((\phi_5 \vee \phi_5') \vee \Delta_5\phi_6 \vee \Delta_5\Delta_6(\phi_7 \vee \phi_7') \vee \cdots \vee \Delta_5\Delta_6\Delta_7\Delta_8\Delta_9\Delta_{10})$$

$$+ (\phi_5\bar{\phi}_5'\phi_6 \vee \phi_5\bar{\phi}_5'\Delta_6(\phi_7 \vee \phi_7') \vee \phi_5\bar{\phi}_5'\Delta_6\Delta_7\phi_8 \vee \cdots \vee \phi_5\bar{\phi}_5'\Delta_6\Delta_7\Delta_8\Delta_9\Delta_{10})$$

$$+ K_{34}\,[\,(\phi_6(\phi_7 \vee \phi_7') \vee \phi_6\Delta_7\phi_8 \vee \phi_6\Delta_7\Delta_8(\phi_9 \vee \phi_9') \vee \cdots \vee \phi_6\Delta_7\Delta_8\Delta_9\Delta_{10})$$

$$+ (\phi_7\bar{\phi}_7'\phi_8 \vee \phi_7\bar{\phi}_7'\Delta_8(\phi_9 \vee \phi_9') \vee \phi_7\bar{\phi}_7'\Delta_8\Delta_9\phi_{10} \vee \phi_7\bar{\phi}_7'\Delta_8\Delta_9\Delta_{10})]$$

$$+ K_{35}\,[\,(\phi_8(\phi_9 \vee \phi_9') \vee \phi_8\Delta_9\phi_{10} \vee \phi_8\Delta_9\Delta_{10})$$

$$+ (\phi_9\bar{\phi}_9'\phi_{10} \vee \phi_9\bar{\phi}_9'\Delta_{10})]$$

$$+ K_{36}\,[\,\phi_{10}]\,\}\; s_o$$

$$z_4(t) = e_6^o + K_{34} [ \quad e'_6 + \lambda_7(e_7^* + e_8^o)]$$

$$+ K_{35} [\lambda_7 e'_8 + \lambda_9(e_9^* + e_{10}^o)]$$

$$\cdot K_{36} [ \lambda_9 e'_{10} + t] \; + z_4^o$$

where

$$z_4^o = \{ (\phi_6 \; \Delta_6( \phi_7 \vee \phi'_7) \vee \Delta_6 \Delta_7 \phi_8 \vee \Delta_6 \Delta_7 \Delta_8 (\phi_9 \vee \phi'_9) \vee \cdots \vee \Delta_6 \Delta_7 \Delta_8 \Delta_9 \Delta_{10})$$

$$+ K_{34} [ (\phi_6(\phi_7 \vee \phi'_7) , \phi_6 \Delta_7 \phi_8 \vee \phi_6 \Delta_7 \Delta_8 (\phi_9 \vee \phi'_9) \vee \cdots \vee \phi_6 \Delta_7 \Delta_8 \Delta_9 \Delta_{10})$$

$$+ (\phi_7 \overline{\phi}'_7 \phi_8 \vee \phi_7 \overline{\phi}'_7 \Delta_8 (\phi_9 \vee \phi'_9) \vee \phi_7 \overline{\phi}'_7 \Delta_8 \Delta_9 \phi_{10} \vee \phi_7 \overline{\phi}'_7 \Delta_8 \Delta_9 \Delta_{10})]$$

$$+ K_{35} [ (\phi_8(\phi_9 \vee \phi'_9) \vee \phi_8 \Delta_9 \; \phi_{10} \vee \phi_8 \Delta_9 \Delta_{10})$$

$$+ (\phi_9 \overline{\phi}'_9 \; \phi_{10} \vee \phi_9 \overline{\phi}'_9 \Delta_{10})]$$

$$+ K_{36} [ \phi_{10}] \} s_o$$

$$z_5(t) = (m_r + 1) F_r + m_r z_2(t)$$

$$z_5^o = m_r z_2^o$$

$$z_6(t) = (m_r+1) F_r + m_r z_4(t)$$

$$z_6^o = m_r z_4^o$$

$$\begin{aligned} z_7(t) &= (m_a+1)\,F_a + m_a\{e_1^* + e_2^o \\ &\quad + K_{12}\,[\ \ e_2' + \lambda_3(e_3^* + e_4^o)] \\ &\quad + K_{13}\,[\lambda_3 e_4' + t]\} + z_7^o \end{aligned}$$

where

$$\begin{aligned} z_7^o = m_a\ \{((&\phi_1 \vee \phi_1') \vee \Delta_1\phi_2 \vee \Delta_1\Delta_2(\phi_3 \vee \phi_3') \vee \Delta_1\Delta_2\Delta_3\phi_4 \\ &\vee \Delta_1\Delta_2\Delta_3\Delta_4(\rho_1 \vee \phi_5 \vee \phi_5') \vee \Delta_1\Delta_2\Delta_3\Delta_4\Delta_5\overline{\rho}_1) \\ &+ (\phi_1\overline{\phi}_1'\phi_2 \vee \phi_1\overline{\phi}_1'\Delta_2(\phi_3 \vee \phi_3') \vee \phi_1\overline{\phi}_1'\Delta_2\Delta_3\phi_4 \\ &\vee \phi_1\overline{\phi}_1'\Delta_2\Delta_3\Delta_4(\rho_1 \vee \phi_5 \vee \phi_5') \vee \phi_1\overline{\phi}_1'\Delta_2\Delta_3\Delta_4\Delta_5\overline{\rho}_1) \end{aligned}$$

$$\begin{aligned} + K_{12}\,[\,&(\phi_2(\phi_3 \vee \phi_3') \vee \phi_2\Delta_3\phi_4 \vee \phi_2\Delta_3\Delta_4(\rho_1 \vee \phi_5 \vee \phi_5') \vee \phi_2\Delta_3\Delta_4\Delta_5\overline{\rho}_1) \\ &+ (\phi_3\overline{\phi}_3'\phi_4 \vee \phi_3\overline{\phi}_3'\Delta_4(\rho_1 \vee \phi_5 \vee \phi_5') \vee \phi_3\overline{\phi}_3'\Delta_4\Delta_5\overline{\rho}_1)] \end{aligned}$$

$$\begin{aligned} + K_{13}\,[\,&(\phi_4(\rho_1 \vee \phi_5 \vee \phi_5') \vee \phi_4\Delta_5\overline{\rho}_1) \\ &+ (\phi_5\overline{\phi}_5'\rho_1)]\}\ s_o \end{aligned}$$

$$z_8(t) = (m_p + 1)\, F_p + m_p \{ e^*_{10} + e^o_9$$

$$+ K_{55} [ \quad e'_9 + \lambda_8 (e^*_8 + e^o_7)$$

$$+ K_{54} [\lambda_8 e'_7 + t]\} + z^o_8$$

where

$$z^o_8 = m_p \{[(( \phi_{10} \vee \phi'_{10}) \vee \Delta_{10} \phi_9 \vee \Delta_{10} \Delta_9 (\phi_8 \vee \phi'_8) \vee \Delta_{10} \Delta_9 \Delta_8 \phi_7$$

$$\vee \Delta_{10} \Delta_9 \Delta_8 \Delta_7 (\rho_3 \vee \phi_6 \vee \phi'_6) \vee \Delta_{10} \Delta_9 \Delta_8 \Delta_7 \Delta_6 \bar{\rho}'_3)$$

$$+ (\phi_{10} \bar{\phi}'_{10}\ \phi_9 \vee \phi_{10} \bar{\phi}'_{10} \Delta_9 (\phi_8 \vee \phi'_8) \vee \phi_{10} \bar{\phi}'_{10} \Delta_9 \Delta_8 \phi_7$$

$$\vee \phi_{10} \bar{\phi}'_{10} \Delta_9 \Delta_8 \Delta_7 (\rho_3 \vee \phi_6 \vee \phi'_6) \vee \phi_{10} \bar{\phi}'_{10} \Delta_9 \Delta_8 \Delta_7 \Delta_6 \bar{\rho}_3)$$

$$+ K_{55} [ (\phi_9 (\phi_8 \vee \phi'_8) \vee \phi_9 \Delta_8 \phi_7 \vee \phi_9 \Delta_8 \Delta_7 (\rho_3 \vee \phi_6 \vee \phi'_6) \vee \phi_9 \Delta_8 \Delta_7 \Delta_6 \bar{\rho}_3)$$

$$+ (\phi_8 \bar{\phi}'_8 \phi_7 \vee \phi_8 \bar{\phi}'_8 \Delta_7 (\rho_3 \vee \phi_6 \vee \phi'_6) \vee \phi_8 \bar{\phi}'_8 \Delta_7 \Delta_6 \bar{\rho}_3)]$$

$$+ K_{54} [ (\phi_7 (\rho_3 \vee \phi_6 \vee \phi'_6) \vee \phi_7 \Delta_6 \bar{\rho}_3)$$

$$+ ( \phi_6 \bar{\phi}'_6 \bar{\rho}_3)] \} s_o$$

$$z_9(t) = (m_a + 1)\,F_a + m_a\{e_1^* + e_2^o$$

$$+ K_{12}\,[\quad e_2' + \lambda_3(e_3^* + e_4^o)]$$

$$+ K_{13}\,[\,\lambda_3 e_4' + \lambda_5\,(e_5^* + e_6^o)]$$

$$+ K_{14}\,[\,\lambda_5 e_6' + \lambda_7(e_7^* + e_8^o)]$$

$$+ K_{15}\,[\,\lambda_7 e_8' + \lambda_9(e_9^* + e_{10}^o)]$$

$$+ K_{16}\,[\,\lambda_9 e_{10}' + t\,]\} + z_9^o$$

where

$$z_9^o = m_a\,\{((\phi_1 \vee \phi_1') \vee \Delta_1\phi_2 \vee \Delta_1\Delta_2(\phi_3 \vee \phi_3') \vee \cdots \vee \Delta_1\Delta_2\Delta_3\Delta_4\Delta_5\Delta_6\Delta_7\Delta_8\Delta_9\Delta_{10})$$
$$+(\phi_1\phi_1'\phi_2 \vee \phi_1\overline{\phi}_1'\Delta_2(\phi_3 \vee \phi_3') \vee \cdots \vee \phi_1\overline{\phi}_1'\Delta_2\Delta_3\Delta_4\Delta_5\Delta_6\Delta_7\Delta_8\Delta_9\Delta_{10})$$

$$+ K_{12}\,[\,(\phi_2(\phi_3 \vee \phi_3') \vee \phi_2\Delta_3\phi_4 \vee \phi_2\Delta_3\Delta_4(\phi_5 \vee \phi_5') \vee \cdots \vee \phi_2\Delta_3\Delta_4\Delta_5\Delta_6\Delta_7\Delta_8\Delta_9\Delta_{10})$$
$$+(\phi_3\overline{\phi}_3'\phi_4 \vee \phi_3\overline{\phi}_3'\Delta_4(\phi_5 \vee \phi_5') \vee \cdots \vee \phi_3\overline{\phi}'_3\Delta_4\Delta_5\Delta_6\Delta_7\Delta_8\Delta_9\Delta_{10})\,]$$

$$+ K_{13}\,[(\phi_4(\phi_5 \vee \phi_5') \vee \phi_4\Delta_5\phi_6 \vee \phi_4\Delta_5\Delta_6(\phi_7 \vee \phi_7') \vee \cdots \vee \phi_4\Delta_5\Delta_6\Delta_7\Delta_8\Delta_9\Delta_{10})$$
$$+(\phi_5\overline{\phi}_5'\phi_6 \vee \phi_5\overline{\phi}_5'\Delta_6(\phi_7 \vee \phi_7') \vee \cdots \vee \phi_5\overline{\phi}_5'\Delta_6\Delta_7\Delta_8\Delta_9\Delta_{10})]$$

$$+ K_{14}\,[\,(\phi_6(\phi_7 \vee \phi_7') \vee \phi_6\Delta_7\,\phi_8 \vee \phi_6\Delta_7\Delta_8(\phi_9 \vee \phi_9') \vee \cdots \vee \phi_6\Delta_7\Delta_8\Delta_9\Delta_{10})$$
$$+(\phi_7\overline{\phi}_7'\phi_8 \vee \phi_7\overline{\phi}_7'\Delta_8(\phi_9 \vee \phi_9') \vee \cdots \vee \phi_7\overline{\phi}_7'\Delta_8\Delta_9\Delta_{10})]$$

$$+ K_{15}\,[\,(\phi_8(\phi_9 \vee \phi_9') \vee \phi_8\Delta_9\phi_{10} \vee \phi_8\Delta_9\Delta_{10})$$
$$+(\phi_9\overline{\phi}_9'\phi_{10} \vee \phi_9\overline{\phi}_9'\Delta_{10})]$$

$$+ K_{16}\,[\,\phi_{10}\,]\}\; s_o$$

$$
\begin{aligned}
z_{10}(t) = (m_p + 1)\,F_p + m_p\{e^{*}_{10} + e^{c}_{9} \\
&+ K_{55}[\quad e'_9 + \lambda_8\,(e^{*}_8 + e^{o}_7)] \\
&+ K_{54}[\,\lambda_8 e'_7 + \lambda_6(e^{*}_6 + e^{o}_5)] \\
&+ K_{53}[\,\lambda_6 e'_5 + \lambda_4(e^{*}_4 + e^{o}_3)] \\
&+ K_{52}[\,\lambda_4 e'_3 + \lambda_2\,(e^{*}_2 + e^{o}_1)] \\
&+ K_{51}[\,\lambda_2 e'_1 + t\,]\} + z^{o}_{10}
\end{aligned}
$$

where

$$
\begin{aligned}
z^{o}_{10} = m_p\,\{((\phi_{10}\vee\phi'_{10})\vee\Delta_{10}\phi_9\vee\Delta_{10}\Delta_9(\phi_8\vee\phi'_8)\vee\cdots\vee\Delta_{10}\Delta_9\Delta_8\Delta_7\Delta_6\Delta_5\Delta_4\Delta_3\Delta_2\Delta_1) \\
&+(\phi_{10}\overline{\phi}'_{10}\phi_9\vee\phi_{10}\overline{\phi}'_{10}\Delta_9(\phi_8\vee\phi'_8)\vee\cdots\vee\phi_{10}\overline{\phi}'_{10}\Delta_9\Delta_8\Delta_7\Delta_6\Delta_5\Delta_4\Delta_3\Delta_2\Delta_1) \\[1ex]
&+ K_{55}\,[\,(\phi_9(\phi_8\vee\phi'_8)\vee\phi_9\Delta_8\phi_7\vee\phi_9\Delta_8\Delta_7\,(\phi_6\vee\phi'_6)\vee\cdots\vee\phi_9\Delta_8\Delta_7\Delta_6\Delta_5\Delta_4\Delta_3\Delta_2\Delta_1) \\
&+(\phi_8\overline{\phi}'_8\phi_7\vee\phi_8\overline{\phi}'_8\Delta_7(\phi_6\vee\phi'_6)\vee\cdots\vee\phi_8\overline{\phi}'_8\Delta_7\Delta_6\Delta_5\Delta_4\Delta_3\Delta_2\Delta_1)] \\[1ex]
&+ K_{54}\,[\,(\phi_7(\phi_6\vee\phi'_6)\vee\phi_7\Delta_6\phi_5\vee\phi_7\Delta_6\Delta_5(\phi_4\vee\phi'_4)\vee\cdots\vee\phi_7\Delta_6\Delta_5\Delta_4\Delta_3\Delta_2\Delta_1) \\
&+(\phi_6\overline{\phi}'_6\phi_5\vee\phi_6\overline{\phi}'_6\Delta_5(\phi_4\vee\phi'_4)\vee\cdots\vee\phi_6\overline{\phi}'_6\Delta_5\Delta_4\Delta_3\Delta_2\Delta_1)] \\[1ex]
&+ K_{53}\,[\,(\phi_5(\phi_4\vee\phi'_4)\vee\phi_5\Delta_4\phi_3\vee\phi_5\Delta_4\Delta_3(\phi_2\vee\phi'_2)\vee\cdots\vee\phi_5\Delta_4\Delta_3\Delta_2\Delta_1) \\
&+(\phi_4\overline{\phi}'_4\phi_3\vee\phi_4\overline{\phi}'_4\Delta_3(\phi_2\vee\phi'_2)\vee\cdots\vee\phi_4\overline{\phi}'_4\Delta_3\Delta_2\Delta_1)] \\[1ex]
&+ K_{52}\,[\,(\phi_3(\phi_2\vee\phi'_2)\vee\phi_3\Delta_2\phi_1\vee\phi_3\Delta_2\Delta_1) \\
&+(\phi_2\overline{\phi}'_2\phi_1\vee\phi_2\overline{\phi}'_2\Delta_1)] \\[1ex]
&+ K_{51}\,[\,\phi_1]\}\,s_o
\end{aligned}
$$

$$
\begin{aligned}
z^*_1(t) &= e^*_6 + e^o_5 + (\hat{K}_{33}-1)\,e'_5 + [\,\lambda_4(\hat{K}_{33}-1) + 1\,]\,(e^*_4 + e^o_3) \\
&+ (\lambda_4\hat{K}_{32} + \bar{\lambda}_4\hat{K}_{22}-1)\,e'_3 + [\,\lambda_2(\hat{K}_{32}-1) + 1\,]\,(e^*_2 + e^o_1) \\
&+ (\lambda_2\hat{K}_{31} + \bar{\lambda}_2\hat{K}_{11}-1)\,e'_1 + \hat{K}_{31}\,t + z^{o*}_1
\end{aligned}
$$

$$
\begin{aligned}
z^*_2(t) &= e^o_5 + (\hat{K}_{33}-1)\,e'_5 + [\,\lambda_4(\hat{K}_{33}-1) + 1\,]\,(e^*_4 + e^o_3) \\
&+ (\lambda_4\hat{K}_{32} + \bar{\lambda}_4\,\hat{K}_{22}-1)\,e'_3 + [\,\lambda_2\,(\hat{K}_{32}-1) + 1\,]\,(e^*_2 + e^o_1) \\
&+ (\lambda_2\hat{K}_{31} + \bar{\lambda}_2\,\hat{K}_{11}-1)\,e'_1 + \hat{K}_{31}\,t + z^{o*}_2
\end{aligned}
$$

$$
\begin{aligned}
z^*_3(t) &= e^*_5 + e^o_6 + (\hat{K}_{34}-1)\,e'_6 + [\,\lambda_7\,(\hat{K}_{34}-1) + 1\,]\,(e^*_7 + e^o_8) \\
&+ (\lambda_7\hat{K}_{35} + \bar{\lambda}_7\,\hat{K}_{45}-1)\,e'_8 + [\,\lambda_9(\hat{K}_{35}-1) + 1\,]\,(e^*_9 + e^o_{10}) \\
&+ (\lambda_9\hat{K}_{36} + \bar{\lambda}_9\,\hat{K}_{56}-1)\,e'_{10} + \hat{K}_{36}\,t + z^{o*}_3
\end{aligned}
$$

$$
\begin{aligned}
z^*_4(t) &= e^o_6 + (\hat{K}_{34}-1)\,e'_6 + [\,\lambda_7(\hat{K}_{34}-1) + 1\,]\,(e^*_7 + e^o_8) \\
&+ (\lambda_7\hat{K}_{35} + \bar{\lambda}_7\,\hat{K}_{45}-1)\,e'_8 + [\,\lambda_9(\hat{K}_{35}-1) + 1\,]\,(e^*_9 + e^o_{10}) \\
&+ (\lambda_9\hat{K}_{36} + \bar{\lambda}_9\,\hat{K}_{56}-1)\,e'_{10} + \hat{K}_{36}\,t + z^{o*}_4
\end{aligned}
$$

The expressions for $z^{o*}_1$, $z^{o*}_2$, $z^{o*}_3$, and $z^{o*}_4$ are the same as those for $z^o_1$, $z^o_2$, $z^o_3$ and $z^o_4$, respectively, with the exception that $K_{ij}$ is replaced by $\hat{K}_{ij}$ wherever it occurs.

$$z^*_5\,(t) = \hat{m}_r\,(F_r + e^o_5) + (\widehat{m_r K_{33}} - 1)\, e'_5 + [\,\lambda_4(\widehat{m_r K_{33}} - 1) + 1]\,(e^*_4 + e^o_3)$$

$$+ (\lambda_4 \widehat{m_r K_{32}} + \bar{\lambda}_4 \hat{K}_{22} - 1)\, e'_3 + [\,\lambda_2(\widehat{m_r K_{32}} - 1) + 1]\,(e^*_2 + e^o_1)$$

$$+ )\lambda_2 \widehat{m_r K_{31}} + \bar{\lambda}_2 \hat{K}_{11} - 1)\, e'_1 + \widehat{m_r K_{31}}\, t + z^{o*}_5$$

$$z^*_6(t) = \hat{m}_r\,(F_r + e^o_6) + (\widehat{m_r K_{34}} - 1) e'_6 + [\,\lambda_7(\widehat{m_r K_{34}} - 1) + 1]\,(e^*_7 + e^o_8)$$

$$+ (\lambda_7 \widehat{m_r K_{35}} + \bar{\lambda}_7 \hat{K}_{45} - 1)\, e'_8 + [\,\lambda_9(\widehat{m_r K_{35}} - 1) + 1]\,(e^*_9 + e^o_{10})$$

$$+ (\lambda_9 \widehat{m_r K_{36}} + \bar{\lambda}_9 \hat{K}_{56} - 1)\, e'_{10} + \widehat{m_r K_{36}}\, t + z^{o*}_6$$

$$z^*_7(t) = \hat{m}_a(F_a + e^*_1 + e^o_2) + (\widehat{m_a K_{12}} - 1)\, e'_2 + [\,\lambda_3(\widehat{m_a K_{12}} - 1) + 1]\,(e^*_3 + e^o_4)$$

$$+ (\lambda_3 \widehat{m_a K_{13}} + \bar{\lambda}_3 \hat{K}_{23} - 1)\, e'_4 + \widehat{m_a k_{13}}\, t + z^{o*}_7$$

$$z^*_8(t) = \hat{m}_p\,(F_p + e^*_{10} + e^o_9) + (\widehat{m_p K_{35}} - 1) e'_9 + [\,\lambda_8(\widehat{m_p K_{35}} - 1) + 1]\,(e^*_8 + e^o_7)$$

$$+ (\lambda_8 \widehat{m_p K_{54}} + \bar{\lambda}_8 \hat{K}_{44} - 1)\, e'_7 + \widehat{m_p K_{54}}\, t + z^{o*}_8$$

$$z^*_9(t) = \hat{m}_a(F_a + e^*_1 + e^o_2) + (\widehat{m_a K_{12}} - 1)\, e'_2 + [\,\lambda_3(\widehat{m_a K_{12}} - 1) + 1]\,(e^*_3 + e^o_4)$$

$$+ (\lambda_3 \widehat{m_a K_{13}} + \bar{\lambda}_3 \hat{K}_{23} - 1)\, e'_4 + [\,\lambda_5(\widehat{m_a K_{13}} - 1) + 1]\,(e^*_5 + e^o_6)$$

$$+ (\lambda_5 \widehat{m_a K_{14}} + \bar{\lambda}_5 \hat{K}_{34} - 1)\, e'_6 + [\,\lambda_7\,(\widehat{m_a K_{14}} - 1) + 1]\,(e^*_7 + e^o_8)$$

$$+ (\lambda_7 \widehat{m_a K_{15}} + \bar{\lambda}_7 \hat{K}_{45} - 1)\, e'_8 + [\,\lambda_9(\widehat{m_a K_{15}} - 1) + 1]\,(e^*_9 + e^o_{10})$$

$$+ (\lambda_9 \widehat{m_a K_{16}} + \bar{\lambda}_9 \hat{K}_{56} - 1)\, e'_{10} + \widehat{m_a K_{16}}\, t + z^{o*}_9$$

$$z^*_{10}(t) = \hat{m}_p(F_p + e^*_{10} + e^o_9) + (\widehat{m_p K_{55}} - 1)\, e'_9 + [\lambda_8(\widehat{m_p K_{55}} - 1) + 1]\,(e^*_8 + e^o_7)$$
$$+ (\lambda_8 \widehat{m_p K_{54}} + \bar{\lambda}_8 \hat{K}_{44} - 1)\, e'_7 + [\lambda_6(\widehat{m_p K_{54}} - 1) + 1]\,(e^*_6 + e^o_5)$$
$$+ (\lambda_6 \widehat{m_p K_{53}} + \bar{\lambda}_6 \hat{K}_{33} - 1)\, e'_5 + [\lambda_4(\widehat{m_p K_{53}} - 1) + 1]\,(e^*_4 + e^o_3)$$
$$+ (\lambda_4 \widehat{m_p K_{52}} + \bar{\lambda}_4 \hat{K}_{22} - 1)\, e'_3 + [\lambda_2(\widehat{m_p K_{52}} - 1) + 1]\,(e^*_2 + e^o_1)$$
$$+ (\lambda_2 \widehat{m_p K_{51}} + \bar{\lambda}_2 \hat{K}_{11} - 1)\, e'_1 + \widehat{m_p K_{51}}\, t + z^{o*}_{10}$$

The expressions for $z^{o*}_5$, $z^{o*}_6$, $z^{o*}_7$, $z^{o*}_8$, $z^{o*}_9$, and $z^{o*}_{10}$ are the same as those for $z^o_5$, $z^o_6$, $z^o_7$, $z^o_8$, $z^o_9$, and $z^o_{10}$, respectively, with the exception that $m_a$, $m_r$, $m_p$ $m_a K_{ij}$, $m_r K_{ij}$, and $m_p K_{ij}$ are replaced by $\hat{m}_a$, $\hat{m}_r$, $\hat{m}_p$, $\widehat{m_a K_{ij}}$, $\widehat{m_r K_{ij}}$, and $\widehat{m_p K_{ij}}$, respectively, wherever they occur.

## Appendix D

## SUMMARY OF TIME COSTS FOR PRIMITIVE OPERATIONS

$$t_{1,1}=T_a+(1-x_1)\,z_7(C_{r_1})$$
$$+x_1(1-x_4)\{(\bar{\Delta}_6+\bar{\Delta}_8)\,z_7^*(C_{r_1})+\Delta_6\Delta_8 z_7(C_{r_1})+m_{r_1}[\rho_1 z_3(C_p)+\bar{\rho}_1 z_4(C_p)]\}$$
$$+x_1x_4\{z_7^*(C_{r_1})+(\hat{m}_{r_1}-1)[\rho_1 z_3(C_p)+\bar{\rho}_1 z_4(C_p)]+[\rho_1 z_3^*(C_p)+\bar{\rho}_1 z_4^*(C_p)]\}$$

$$t_{1,2}=T_r+(1-x_1)z_5(C_a)$$
$$+x_1(1-x_4)[(\bar{\Delta}_6+\bar{\Delta}_8)z_5^*(C_a)+\Delta_6\Delta_8 z_5(C_a)+m_{r_1}z_4(C_p)]$$
$$+x_1x_4[z_5^*(C_a)+(\hat{m}_{r_1}-1)z_4(C_p)+z_4^*(C_p)]$$

$$t_{1,3}=T_r+(1-x_2)z_6(C_p)$$
$$+x_2(1-x_5)[\delta_p(\bar{\Delta}_3+\bar{\Delta}_5)z_6^*(C_p)+(\bar{\delta}_p+\Delta_3\Delta_5)z_6(C_p)+m_{z_p}z_2(C_a)]$$
$$+x_2x_5[z_6^*(C_p)+(\hat{m}_{z_p}-1)z_2(C_a)+z_2^*(C_a)]$$

$$t_{1,4}=T_p+(1-x_2)z_8(C_{r_2})$$
$$+x_2(1-x_5)\{\delta_p(\bar{\Delta}_3+\bar{\Delta}_5)z_8^*(C_{r_2})+(\bar{\delta}_p+\Delta_3\Delta_5)z_8(C_{r_2})+m_{z_p}[\rho_3 z_1(C_a)+\bar{\rho}_3 z_2(C_a)]\}$$
$$+x_2x_5\{z_8^*(C_{r_2})+(\hat{m}_{z_p}-1)[\rho_3 z_1(C_a)+\bar{\rho}_3 z_2(C_a)]+[\rho_3 z_1^*(C_a)+\bar{\rho}_3 z_2^*(C_a)]\}$$

$$t_{2,1} = T_a + (1-x_1) z_7(C_{r_1}) + x_1 z_7^*(C_{r_1})$$

$$t_{2,2} = T_r + (1-x_1) z_5(C_a) + x_1 z_5^*(C_a)$$

$$t_{3,1} = T_a + (\bar{\Delta}_6 + \bar{\Delta}_8) z_7^*(C_{r_1}) + \Delta_6 \Delta_8 z_7(C_{r_1}) + m_{r_1} [\rho_1 z_3(V_p) + \bar{\rho}_1 z_4(V_p)]$$

$$t_{3,2} = T_r + (\bar{\Delta}_6 + \bar{\Delta}_8) z_5^*(C_a) + \Delta_6 \Delta_8 z_5(C_a) + m_{r_1} z_4(V_p)$$

$$t_{3,3} = k_p^O T_p + k_p^O \{(1-x_2) z_8(C_{r_2})$$

$$+ x_2(1-x_5)\{\delta_p(\bar{\Delta}_3 + \bar{\Delta}_5) z_8^*(C_{r_2}) + (\bar{\delta}_p + \Delta_3 \Delta_5) z_8(C_{r_2}) + m_{z_p} [\rho_3 z_1(C_a) + \bar{\rho}_3 z_2(C_a)]\}$$

$$+ x_2 x_5 \{z_8^*(C_{r_2}) + (\hat{m}_{z_p} - 1)[\rho_3 z_1(C_a) + \bar{\rho}_3 z_2(C_a)] + [\rho_3 z_1^*(C_a) + \bar{\rho}_3 z_2^*(C_a)]\}\}$$

$$+ m_{r_1} K_{36} V_p$$

$$= k_p^O t_{1,4} + m_{r_1} K_{36} V_p$$

$$t_{4,1} = T_r + (1-x_2) z_6(C_p) + x_2 z_6^*(C_p)$$

$$t_{4,2} = T_p + (1-x_2) z_8(C_{r_2}) + x_2 z_8^*(C_{r_2})$$

$$t_{5,1} = T_r + \delta_p(\bar{\Delta}_3 + \bar{\Delta}_5) z_6^*(C_p) + (\bar{\delta}_p + \Delta_3 \Delta_5) z_6(C_p) + m_{z_p} z_2(V_a)$$

$$t_{5,2} = T_p + \delta_p(\bar{\Delta}_3 + \bar{\Delta}_5) z_8^*(C_{r_2}) + (\bar{\delta}_p + \Delta_3 \Delta_5) z_8(C_{r_2}) + m_{z_p}[\rho_3 z_1(V_a) + \bar{\rho}_3 z_2(V_a)]$$

$$\begin{aligned}
t_{5,3} = {} & k_a^o T_a + k_a^o \{(1-x_1) z_7(C_{r_1}) \\
& + x_1(1-x_4)\{(\bar{\Delta}_6 + \bar{\Delta}_8) z_7^*(C_{r_1}) + \Delta_6 \Delta_8 z_7(C_{r_1}) + m_{r_1}[\rho_1 z_3(C_p) + \bar{\rho}_1 z_4(C_p)]\} \\
& + x_1 x_4 \{z_7^*(C_{r_1}) + (\hat{m}_{r_1} - 1)[\rho_1 z_3(C_p) + \bar{\rho}_1 z_4(C_p)] + [\rho_1 z_3^*(C_p) + \bar{\rho}_1 z_4^*(C_p)]\}\} \\
& + m_{z_p} K_{31} V_a \\
= {} & k_a^o t_{1,1} + m_{z_p} K_{31} V_a
\end{aligned}$$

$$t_{6,1} = T_a + (1-x_7) z_9(C_p) + x_7 z_9^*(C_p)$$

$$t_{6,2} = T_p + (1-x_7) z_{10}(C_a) + x_7 z_{10}^*(C_a)$$

$$\begin{aligned}
t_{7,1} = {} & T_a + z_7(V_{r_1}) + m_a K_{13}\{(1-x_4)[\rho_1 z_3(C_p) + \bar{\rho}_1 z_4(C_p) - v_r] \\
& + x_4[\rho_1 z_3^*(C_p) + \bar{\rho}_1 z_4^*(C_p)]\}
\end{aligned}$$

$$\begin{aligned}
t_{7,2} = {} & T_p + z_8(V_{r_2}) + m_p K_{54}\{(1-x_5)[\rho_3 z_1(C_a) + \bar{\rho}_3 z_2(C_a) - v_r] \\
& + x_5[\rho_3 z_1^*(C_a) + \bar{\rho}_3 z_2^*(C_a)]\}
\end{aligned}$$

$$t_{7,3} = k_r^o\, T_r + k_r^o \{(1-x_1) z_5(C_a)$$

$$+x_1(1-x_4)[\, (\bar{\Delta}_6+\bar{\Delta}_8) z_5^*(C_a) + \Delta_6 \Delta_8 z_5(C_a) + m_{r_1} z_4(C_p)]$$

$$+x_1 x_4\, [\, z_5^*(C_a) + (\hat{m}_{r_1} - 1) z_4(C_p) + z_4^*(C_p) + v_r] \}$$

$$= k_r^o\, t_{1,2} + x_1 x_4 k_r^o\, v_r$$

$$t_{7,4} = k_r^o\, T_r + k_r^o\, \{(1-x_2) z_6(C_p)$$

$$+x_2(1-x_5)[\, \delta_p(\bar{\Delta}_3 + \bar{\Delta}_5) z_6^*\, (C_p) + (\bar{\delta}_p + \Delta_3 \Delta_5) z_6(C_p) + m_{z_p} z_2(C_a)]$$

$$+x_2 x_5[\, z_6^*(C_p) + (\hat{m}_{z_p} - 1) z_2(C_a) + z_2^*\, (C_a) + v_r] \}$$

$$= k_r^o\, t_{1,3} + x_2 x_5 k_r^o\, v_r$$

$$t_{8,1} = T_r + z_5(V_a)$$

$$t_{8,2} = k_a^o\, T_a + k_a^o\, \{(1-x_1) z_7(C_{r_1}) + x_1[\, z_7^*\, (C_{r_1}) + V_a] \}$$

$$= k_a^o\, t_{2,1} + x_1\, k_a^o\, V_a$$

$$t_{9,1} = T_r + z_6(V_p)$$

$$t_{9,2} = k_p^o T_p + k_p^o \{(1-x_2) z_8(C_{r_2}) + x_2[\, z_8^*(C_{r_2}) + V_p]\}$$

$$= k_p^o\, t_{4,2} + x_2 k_p^o V_p$$

$$t_{10,1} = T_r + z_5(V_a) + m_r K_{31} z_4(V_p)$$

$$t_{10,2} = T_r + z_6(V_p) + m_r K_{36} z_2(V_a)$$

$$t_{10,3} = k_a^o T_a + k_a^o \{(1-x_1) z_7(C_{r_1}) + x_1[\, z_7^*(C_{r_1}) + V_a + \rho_1 z_3(V_p) + \bar{\rho}_1 z_4(V_p)]\}$$

$$t_{10,4} = k_p^o T_p + k_p^o \{(1-x_2) z_8(C_{r_2}) + x_2[\, z_8^*(C_{r_2}) + V_p + \rho_3 z_1(V_a) + \bar{\rho}_3\, z_2(V_a)]\}$$

$$t_{11,1} = T_a + z_7(V_{r_1})$$

$$t_{11,2} = k_r^o\, T_r + k_r^o \{(1-x_1) z_5(C_a) + x_1[\, z_5^*(C_a) + v_r]\}$$

$$t_{12,1} = T_a + z_9(V_p)$$

$$t_{12,2} = k_p^O T_p + k_p^O \{(1-x_7) z_{10}(C_a) + z_7 [ z_{10}^*(C_a) + V_p ] \}$$

$$t_{13,1} = T_a + z_9(V_p) + k_2^O k_4^O V_{r_1}$$

$$t_{13,2} = k_r^O T_r + k_r^O \{(1-x_1) z_5(C_a) + x_1 [ z_5^*(C_a) + v_r + z_4(V_p) ] \}$$

$$t_{14,1} = T_p + z_{10}(V_a)$$

$$t_{14,2} = k_a^O T_a + k_a^O \{(1-x_7) z_9 (C_p) + x_7 [ z_9^*(C_p) + V_a ] \}$$

$$t_{15,1} = T_p + z_8(V_{r_2})$$

$$t_{15,2} = k_r^O T_r + k_r^O \{(1-x_2) z_6(C_p) + x_2 [ z_6^* (C_p) + v_r ] \}$$

$$t_{16,1} = T_p + z_{10}(V_a) + (k_7^O \ k_9^O / m_{r_p}) V_{r_2}$$

$$t_{16,2} = k_r^O T_r + k_r^O \{(1-x_2) z_6(C_p) + x_2 [ z_6^* (C_p) + v_r + z_2(V_a) ] \}$$

# Appendix E

## SOLUTION LISTING FOR CASE 1 OF THE MEDICAL DIAGNOSIS PROBLEM

In order to interpret the notation used in listing the best solutions within each $\phi$-family, the following comments may be helpful.

The ten digits following a character string of the form

**PF(I):

represent the values of $\phi_1 \ldots \phi_{10}$ for $\phi$-family I, where the value of I simply indicates the position of the $\phi$-family in the sequence of $\phi$-families generated and is used to simplify distinguishing the various $\phi$-families when referring to them.

The number associated with COST is the value of the time cost function for the best solution (or solutions) within the given $\phi$-family, and the number associated with CPU reflects the cummulative CPU time in seconds used to that point in the solution process.

Following the line containing COST and CPU for a given $\phi$-family are one or more pairs of lines representing the best solution (or solutions, if there is more than one pair of these lines) within the $\phi$-family. The first line of the pair contains values for $\phi_1 \ldots \phi_{10}, \phi'_1 \ldots \phi'_{10}$, $\Delta_1 \ldots \Delta_{10}$, and $\beta_2 \ldots \beta_{10}$ in that order, followed by the symbol S and a number which represents the value[*] of the storage cost function for the given solution. The values of $\phi_1 \ldots \phi_{10}$ are the same as those indicated for the $\phi$-family, of course. The second line of the pair contains sixteen digits, each of which represents the method chosen for one of the sixteen primitive operations, where a value of zero indicates that the corresponding operation is not under consideration.

Following the last pair of solution lines is a line containing the entries N and R. The number associated with N indicates the number of feasible solutions contained in the $\phi$-family and the number associated with R indicates the number of those solutions rejected because they violate the given storage constraint.

Finally, appearing at the very end of the solution listing are a number of summary items. The number of the $\phi$-family containing the overall optimal solution is given here, along with the total number of feasible solutions considered in the solution process. ([illegible]e that if two or more $\phi$-families contain solutions which are considered optimal, the number of the $\phi$-family given as containing the optimal solution(s) is that of the last one encountered in the sequence of $\phi$-families.) Also given in this summary are the average and the maximum values of the time cost function over all feasible solutions.

[*] The actual value of the storage cost function is 0.5 units less than the value shown here. 0.5 is added to the actual value to effect round-off when this value is printed as an integer.

```
********** SOLUTION BEGINS **********

**PF( 1): 0000000000

  COST: 0.27291162E 03   CPU:   4.080
  0000000000 0001010000 1110101111 110000111  S: 0.30686500E 05
  0010001000010100
  N:  64  R:   0

**PF( 2): 0000000010

  COST: 0.31865479E 03   CPU:   7.994
  0000000010 0001000010 1110101100 110000100  S: 0.15544500E 05
  0010001000010100
  N:  64  R:   0

**PF( 3): 0000000100

  COST: 0.29781177E 03   CPU:  11.900
  0000000100 0100000100 1011111010 001101000  S: 0.32042500E 05
  0010001000010100
  0000000100 0001000100 1110111010 110001000  S: 0.32042500E 05
  0010001000010100
  N:  64  R:   0

**PF( 4): 0000001000

  COST: 0.25741382E 03   CPU:  17.653
  0000001000 0000001100 1011110011 001100001  S: 0.33398500E 05
  0010001000010100
  0000001000 0000001100 1110110011 110000001  S: 0.33398500E 05
  0010001000010100
  N:  96  R:   0

**PF( 5): 0000001010

  COST: 0.31881689E 03   CPU:  21.508
  0000001010 0000001010 1011110000 001100000  S: 0.18256500E 05
  0010001000010100
  0000001010 0000001010 1110110000 110000000  S: 0.18256500E 05
  0010001000010100
  N:  64  R:   0

**PF( 6): 0000001100

  COST: 0.30840747E 03   CPU:  25.470
  0000001100 0000001100 1011110010 001100000  S: 0.34754500E 05
  0010001000010100
  0000001100 0000001100 1110110010 110000000  S: 0.34754500E 05
  0010001000010100
  N:  64  R:   0
```

```
**PF( 7): 0000010000

  COST: 0.26728027E 03   CPU:  31.347
  0000010000 0100010100 1011101011 001100001  S: 0.34754500E 05
  0010001000010100
  0000010000 0001010100 1110101011 110000001  S: 0.34754500E 05
  0010001000010100
  N:  96  R:   0

**PF( 8): 0000010010

  COST: 0.32868311E 03   CPU:  35.200
  0000010010 0100010010 1011101000 001100000  S: 0.19612500E 05
  0010001000010100
  0000010010 0001010010 1110101000 110000000  S: 0.19612500E 05
  0010001000010100
  N:  64  R:   0

**PF( 9): 0000010100

  COST: 0.32350000E 03   CPU:  39.066
  0000010100 0100010100 1011101010 001100000  S: 0.36110500E 05
  0010001000010100
  0000010100 0001010100 1110101010 110000000  S: 0.36110500E 05
  0010001000010100
  N:  64  R:   0

**PF(10): 0000100000

  COST: 0.25296194E 03   CPU:  46.690
  0000100000 0000110000 1011001111 001000111  S: 0.33398500E 05
  0010001000010100
  0000100000 0000100100 1011011011 001001001  S: 0.33398500E 05
  0010001000010100
  0000100000 0000110000 1110001111 110000111  S: 0.33398500E 05
  0010001000010100
  0000100000 0000100100 1110011011 110001001  S: 0.33398500E 05
  0010001000010100
  N: 128  R:   0

**PF(11): 0000100010

  COST: 0.31436475E 03   CPU:  52.439
  0000100010 0000100010 1011001100 001000100  S: 0.18256500E 05
  0010001000010100
  0000100010 0000100010 1011011000 001001000  S: 0.18256500E 05
  0010001000010100
  0000100010 0000100010 1110001100 110000100  S: 0.18256500E 05
  0010001000010100
  0000100010 0000100010 1110011000 110001000  S: 0.18256500E 05
  0010001000010100
  N:  96  R:   0
```

```
**PF(12): 0000100100

  COST: 0.30917554E 03   CPU:  58.208
  0000100100 0000100100 1011011010 001001000  S: 0.34754500E 05
  0010001000010100
  0000100100 0000100100 1110011010 110001000  S: 0.34754500E 05
  0010001000010100
  N:  96  R:   0

**PF(13): 0000101000

  COST: 0.26881396E 03   CPU:  63.894
  0000101000 0000101100 1011010011 001000001  S: 0.37466500E 05
  0010001000010100
  0000101000 0000101100 1110010011 110000001  S: 0.37466500E 05
  0010001000010100
  N:  96  R:   0

**PF(14): 0000101010

  COST: 0.33021655E 03   CPU:  67.711
  0000101010 0000101010 1011010000 001000000  S: 0.22324500E 05
  0010001000010100
  0000101010 0000101010 1110010000 110000000  S: 0.22324500E 05
  0010001000010100
  N:  64  R:   0

**PF(15): 0000101100

  COST: 0.31980786E 03   CPU:  71.573
  0000101100 0000101100 1011010010 001000000  S: 0.38822500E 05
  0010001000010100
  0000101100 0000101100 1110010010 110000000  S: 0.38822500E 05
  0010001000010100
  N:  64  R:   0

**PF(16): 0000110000

  COST: 0.27864404E 03   CPU:  77.432
  0000110000 0000110100 1011001011 001000001  S: 0.37466500E 05
  0010001000010100
  0000110000 0000110100 1110001011 110000001  S: 0.37466500E 05
  0010001000010100
  N:  96  R:   0

**PF(17): 0000110010

  COST: 0.34004712E 03   CPU:  81.256
  0000110010 0000110010 1011001000 001000000  S: 0.22324500E 05
  0010001000C'0100
  0000110010 0000110010 1110001000 110000000  S: 0.22324500F 05
  0010001000010100
  N:  64  R:   0
```

```
**PF(18): 0000110100

  COST: 0.33486401E 03   CPU:  85.122
  0000110100 0000110100 1011001010 001000000  S: 0.38822500E 05
  0010001000010100
  0000110100 0000110100 1110001010 110000000  S: 0.38822500E 05
  0010001000010100
  N:  64  R:   0

**PF(19): 0001000000

  COST: 0.25642383E 03   CPU:  96.713
  0001000000 0001010000 1110101111 110000111  S: 0.32494500E 05
  0010001000010100
  0001000000 0001000100 1110111011 110001001  S: 0.32494500E 05
  0010001000010100
  N: 192  R:   0

**PF(20): 0001000010

  COST: 0.31782666E 03   CPU: 105.245
  0001000010 0001000010 1110101100 110000100  S: 0.17352500E 05
  0010001000010100
  0001000010 0001000010 1110111000 110001000  S: 0.17352500E 05
  0010001000010100
  N: 144  R:   0

**PF(21): 0001000100

  COST: 0.31264380E 03   CPU: 113.689
  0001000100 0001000100 1110111010 110001000  S: 0.33850500E 05
  0010001000010100
  N: 144  R:   0

**PF(22): 0001001000

  COST: 0.27228174E 03   CPU: 122.251
  0001001000 0001001100 1110110011 110000001  S: 0.36562500E 05
  0010001000010100
  N: 144  R:   0

**PF(23): 0001001010

  COST: 0.33368457E 03   CPU: 127.802
  0001001010 0001001010 1110110000 110000000  S: 0.21420500E 05
  0010001000010100
  N:  96  R:   0

**PF(24): 0001001100

  COST: 0.32327588E 03   CPU: 133.406
  0001001100 0001001100 1110110010 110000000  S: 0.37918500E 05
  0010001000010100
  N:  96  R:   0
```

```
**PF(25): 0001010000

  COST: 0.28211206E 03   CPU: 141.994
  0001010000 0001010100 1110101011 110000001  S: 0.36562500E 05
  001000100001010O
  N: 144  R:   0

**PF(26): 0001010010

  COST: 0.34351489E 03   CPU: 147.505
  0001010010 0001010010 1110101000 110000000  S: 0.21420500E 05
  0010001000010100
  N:  96  R:   0

**PF(27): 0001010100

  COST: 0.33833203E 03   CPU: 153.197
  0001010100 0001010100 1110101010 110000000  S: 0.37918500E 05
  0010001000010100
  N:  96  R:   0

**PF(28): 0010000000

  COST: 0.25296194E 03   CPU: 159.030
  0010000000 0010010000 1001101111 000100111  S: 0.33398500E 05
  0010001000010100
  0010000000 0010000100 1001111011 000101001  S: 0.33398500E 05
  0010001000010100
  N:  96  R:   0

**PF(29): 0010000010

  COST: 0.31436475E 03   CPU: 163.814
  0010000010 0010000010 1001101100 000100100  S: 0.18256500E 05
  0010001000010100
  0010000010 0010000010 1001111000 000101000  S: 0.18256500E 05
  0010001000010100
  N:  80  R:   0

**PF(30): 0010000100

  COST: 0.30398584E 03   CPU: 168.564
  0010000100 0011000100 1100111010 100001000  S: 0.33398500E 05
  0010001000010100
  N:  80  R:   0

**PF(31): 0010001000

  COST: 0.26358789E 03   CPU: 174.214
  0010001000 0010001100 1100110011 100000001  S: 0.34754500E 05
  0010001000010100
  N:  96  R:   0
```

```
**PF(32): 0010001010

  COST: 0.32499072E 03   CPU: 177.958
  0010001010 0010001010 1100110000 100000000  S: 0.19612500E 05
  0010001000010100
  N:  64  R:   0

**PF(33): 0010001100

  COST: 0.31458154E 03   CPU: 181.806
  0010001100 0010001100 1100110010 100000000  S: 0.36110500E 05
  0010001000010100
  N:  64  R:   0

**PF(34): 0010010000

  COST: 0.27345410E 03   CPU: 187.486
  0010010000 0011010100 1100101011 100000001  S: 0.36110500E 05
  0010001000010100
  N:  96  R:   0

**PF(35): 0010010010

  COST: 0.33485693E 03   CPU: 191.280
  0010010010 0011010010 1100101000 100000000  S: 0.20968500E 05
  0010001000010100
  N:  64  R:   0

**PF(36): 0010010100

  COST: 0.32967383E 03   CPU: 195.043
  0010010100 0011010100 1100101010 100000000  S: 0.37466500E 05
  0010001000010100
  N:  64  R:   0

**PF(37): 0010100000

  COST: 0.25913574E 03   CPU: 200.745
  0010100000 0010110000 1100001111 100000111  S: 0.34754500E 05
  0010001000010100
  0010100000 0010100100 1100011011 100001001  S: 0.34754500E 05
  0010001000010100
  N:  96  R:   0

**PF(38): 0010100010

  COST: 0.32053857E 03   CPU: 204.995
  0010100010 0010100010 1100001100 100000100  S: 0.19612500E 05
  0010001000010100
  0010100010 0010100010 1100011000 100001000  S: 0.19612500E 05
  0010001000010100
  N:  72  R:   0
```

```
**PF(39): 0010100100

   COST: 0.31534961E 03   CPU: 209.299
   0010100100 0010100100 1100011010 100001000  S: 0.36110500E 05
   0010001000010100
   N:  72  R:   0

**PF(40): 0010101000

   COST: 0.27498779E 03   CPU: 213.573
   0010101000 0010101100 1100010011 100000001  S: 0.38822500E 05
   0010001000010100
   N:  72  R:   0

**PF(41): 0010101010

   COST: 0.33639062E 03   CPU: 216.410
   0010101010 0010101010 1100010000 100000000  S: 0.23680500E 05
   0010001000010100
   N:  48  R:   0

**PF(42): 0010101100

   COST: 0.32598169E 03   CPU: 219.277
   0010101100 0010101100 1100010010 100000000  S: 0.40178500E 05
   0010001000010100
   N:  48  R:   0

**PF(43): 0010110000

   COST: 0.28481787E 03   CPU: 223.520
   0010110000 0010110100 1100001011 100000001  S: 0.38822500E 05
   0010001000010100
   N:  72  R:   0

**PF(44): 0010110010

   COST: 0.34622095E 03   CPU: 226.373
   0010110010 0010110010 1100001000 100000000  S: 0.23680500E 05
   0010001000010100
   N:  48  R:   0

**PF(45): 0010110100

   COST: 0.34103809E 03   CPU: 229.201
   0010110100 0010110100 1100001010 100000000  S: 0.40178500E 05
   0010001000010100
   N:  48  R:   0
```

```
**PF(46): 0011000000

  COST: 0.26059961E 03   CPU: 236.847
  0011000000 0011010000 1100101111 100000111  S: 0.33850500E 05
  0010001000010100
  0011000000 0011000100 1100111011 100001001  S: 0.33850500E 05
  0010001000010100
  N: 128  R:   0

**PF(47): 0011000010

  COST: 0.32200244E 03   CPU: 242.586
  0011000010 0011000010 1100101100 100000100  S: 0.18708500E 05
  0010001000010100
  0011000010 0011000010 1100111000 100001000  S: 0.18708500E 05
  0010001000010100
  N:  96  R:   0

**PF(48): 0011000100

  COST: 0.31681982E 03   CPU: 248.252
  0011000100 0011000100 1100111010 100001000  S: 0.35206500E 05
  0010001000010100
  N:  96  R:   0

**PF(49): 0011001000

  COST: 0.27645776E 03   CPU: 253.958
  0011001000 0011001100 1100110011 100000001  S: 0.37918500E 05
  0010001000010100
  N:  96  R:   0

**PF(50): 0011001010

  COST: 0.33786060E 03   CPU: 257.620
  0011001010 0011001010 1100110000 100000000  S: 0.22776500E 05
  0010001000010100
  N:  64  R:   0

**PF(51): 0011001100

  COST: 0.32745166E 03   CPU: 261.398
  0011001100 0011001100 1100110010 100000000  S: 0.39274500E 05
  0010001000010100
  N:  64  R:   0

**PF(52): 0011010000

  COST: 0.28628809E 03   CPU: 267.116
  0011010000 0011010100 1100101011 100000001  S: 0.37918500E 05
  0010001000010100
  N:  96  R:   0
```

```
**PF(53): 0011010010

  COST: 0.34769116E 03   CPU: 270.921
  0011010010 0011010010 1100101000 100000000  S: 0.22776500E 05
  0010001000010100
  N:  64  R:   0

**PF(54): 0011010100

  COST: 0.34250806E 03   CPU: 274.643
  0011010100 0011010100 1100101010 100000000  S: 0.39274500E 05
  0010001000010100
  N:  64  R:   0

**PF(55): 0100000000

  COST: 0.25642383E 03   CPU: 284.501
  0100000000 0100010000 1011101111 001100111  S: 0.32494500E 05
  0010001000010100
  0100000000 0100000100 1011111011 001101001  S: 0.32494500E 05
  0010001000010100
  N: 160  R:   0

**PF(56): 0100000010

  COST: 0.31782666E 03   CPU: 292.119
  0100000010 0100000010 1011101100 001100100  S: 0.17352500E 05
  0010001000010100
  0100000010 0100000010 1011111000 001101000  S: 0.17352500E 05
  0010001000010100
  N: 128  R:   0

**PF(57): 0100000100

  COST: 0.30710547E 03   CPU: 299.813
  0100000100 0101000100 1010111010 000001000  S: 0.33398500E 05
  0010001000010100
  N: 128  R:   0

**PF(58): 0100001000

  COST: 0.26670776E 03   CPU: 308.275
  0100001000 0100001100 1010110011 000000001  S: 0.34754500E 05
  0010001000010100
  N: 144  R:   0

**PF(59): 0100001010

  COST: 0.32811060E 03   CPU: 313.845
  0100001010 0100001010 1010110000 000000000  S: 0.19612500E 05
  0010001000010100
  N:  96  R:   0
```

```
**PF(60): 0100001100

  COST: 0.31770166E 03   CPU: 319.521
  0100001100 0100001100 1010110010 000000000  S: 0.36110500E 05
  0010001000010100
  N:  96  R:   0

**PF(61): 0100010000

  COST: 0.27657422E 03   CPU: 327.953
  0100010000 0101010100 1010101011 000000001  S: 0.36110500E 05
  0010001000010100
  N: 144  R:   0

**PF(62): 0100010010

  COST: 0.33797705E 03   CPU: 333.529
  0100010010 0101010010 1010101000 000000000  S: 0.20968500E 05
  0010001000010100
  N:  96  R:   0

**PF(63): 0100010100

  COST: 0.33279395E 03   CPU: 339.118
  0100010100 0101010100 1010101010 000000000  S: 0.37466500E 05
  0010001000010100
  N:  96  R:   0

**PF(64): 0100100000

  COST: 0.26225562E 03   CPU: 346.644
  0100100000 0100110000 1010001111 000000111  S: 0.34754500E 05
  0010001000010100
  0100100000 0100100100 1010011011 000001001  S: 0.34754500E 05
  0010001000010100
  N: 128  R:   0

**PF(65): 0100100010

  COST: 0.32365845E 03   CPU: 352.244
  0100100010 0100100010 1010001100 000000100  S: 0.19612500E 05
  0010001000010100
  0100100010 0100100010 1010011000 000001000  S: 0.19612500E 05
  0010001000010100
  N:  96  R:   0

**PF(66): 0100100100

  COST: 0.31846973E 03   CPU: 357.780
  0100100100 0100100100 1010011010 000001000  S: 0.36110500E 05
  0010001000010100
  N:  96  R:   0
```

```
**PF(67): 0100101000

  COST: 0.27810791E 03   CPU: 363.331
  0100101000 0100101100 1010010011 000000001  S: 0.38822500E 05
  0010001000010100
  N:  96  R:    0

**PF(68): 0100101010

  COST: 0.33951074E 03   CPU: 367.008
  0100101010 0100101010 1010010000 000000000  S: 0.23680500E 05
  0010001000010100
  N:  64  R:    0

**PF(69): 0100101100

  COST: 0.32910132E 03   CPU: 370.736
  0100101100 0100101100 1010010010 000000000  S: 0.40178500E 05
  0010001000010100
  N:  64  R:    0

**PF(70): 0100110000

  COST: 0.28793799E 03   CPU: 376.424
  0100110000 0100110100 1010001011 000000001  S: 0.38822500E 05
  0010001000010100
  N:  96  R:    0

**PF(71): 0100110010

  COST: 0.34934131E 03   CPU: 380.186
  0100110010 0100110010 1010001000 000000000  S: 0.23680500E 05
  0010001000010100
  N:  64  R:    0

**PF(72): 0100110100

  COST: 0.34415796E 03   CPU: 383.873
  0100110100 0100110100 1010001010 000000000  S: 0.40178500E 05
  0010001000010100
  N:  64  R:    0

**PF(73): 0101000000

  COST: 0.26571729E 03   CPU: 391.424
  0101000000 0101010000 1010101111 000000111  S: 0.33850500E 05
  0010001000010100
  0101000000 0101000100 1010111011 000001001  S: 0.33850500E 05
  0010001000010100
  N: 128  R:    0
```

```
**PF(74): 0101000010

  COST: 0.32712061E 03   CPU: 397.076
  0101000010 0101000010 1010101100 000000100  S: 0.18708500E 05
  0010001000010100
  0101000010 0101000010 1010111000 000001000  S: 0.18708500E 05
  0010001000010100
  N:  96  R:   0

**PF(75): 0101000100

  COST: 0.32193750E 03   CPU: 402.683
  0101000100 0101000100 1010111010 000001000  S: 0.35206500E 05
  0010001000010100
  N:  96  R:   0

**PF(76): 0101001000

  COST: 0.28157593E 03   CPU: 408.477
  0101001000 0101001100 1010110011 000000001  S: 0.37918500E 05
  0010001000010100
  N:  96  R:   0

**PF(77): 0101001010

  COST: 0.34297852E 03   CPU: 412.182
  0101001010 0101001010 1010110000 000000000  S: 0.22776500E 05
  0010001000010100
  N:  64  R:   0

**PF(78): 0101001100

  COST: 0.33256958E 03   CPU: 415.920
  0101001100 0101001100 1010110010 000000000  S: 0.39274500E 05
  0010001000010100
  N:  64  R:   0

**PF(79): 0101010000

  COST: 0.29140576E 03   CPU: 421.525
  0101010000 0101010100 1010101011 000000001  S: 0.37918500E 05
  0010001000010100
  N:  96  R:   0

**PF(80): 0101010010

  COST: 0.35280908E 03   CPU: 425.174
  0101010010 0101010010 1010101000 000000000  S: 0.22776500E 05
  0010001000010100
  N:  64  R:   0
```

```
**PF(81): 0101010100

  COST: 0.34762622E 03   CPU: 428.804
  0101010100 0101010100 1010101010 000000000  S: 0.39274500E 05
  0010001000010100
  N:  64  R:   0

PF(28) CONTAINS BEST SOLUTION(S).   7232 SOLUTIONS CONSIDERED.

   MEAN COST OVER ALL SOLUTIONS:    0.35064868E 03
   MAXIMUM COST ENCOUNTERED:        0.57534351E 03

********** SOLUTION TERMINATED *********
```

# Appendix F

## SOLUTION LISTING FOR CASE 2

## OF THE MEDICAL DIAGNOSIS PROBLEM

```
********** SOLUTION BEGINS **********

**PF( 1): 0000000000

   COST: 0.19451155E 03   CPU:   4.286
   0000000000 0001000000 1110101111 110000111  S: 0.18482500E 05
   0010001000010100
   N:  64  R:   0

**PF( 2): 0000000010

   COST: 0.23521910E 03   CPU:   8.485
   0000000010 0001000010 1110101101 110000100  S: 0.19838500E 05
   0010001000010100
   N:  64  R:   0

**PF( 3): 0000000100

   COST: 0.22721243E 03   CPU:  12.602
   0000000100 0100000110 1011111001 001101000  S: 0.26618500E 05
   0010002000010100
   0000000100 0001000110 1110111001 110001000  S: 0.26618500E 05
   0010002000010100
   N:  64  R:   0

**PF( 4): 0000001000

   COST: 0.19406752E 03   CPU:  18.755
   0000001000 0000001000 1011110011 001100001  S: 0.22098500E 05
   0010001000010100
   0000001000 0000001000 1110110011 110000001  S: 0.22098500E 05
   0010001000010100
   N:  96  R:   0
```

```
**PF( 5): 0000001010

  COST: 0.23477510E 03   CPU:  22.885
  0000001010 0000001010 1011110001 001100000  S: 0.23454500E 05
  0010001000010100
  0000001010 0000001010 1110110001 110000000  S: 0.23454500E 05
  0010001000010100
  N:  64  R:   0

**PF( 6): 0000001100

  COST: 0.24145790E 03   CPU:  27.060
  0000001100 0000001100 1011110011 001100001  S: 0.34754500E 05
  0010001000010100
  0000001100 0000001100 1110110011 110000001  S: 0.34754500E 05
  0010001000010100
  N:  64  R:   0

**PF( 7): 0000010000

  COST: 0.20734515E 03   CPU:  33.207
  0000010000 0100010000 1011101011 001100001  S: 0.23906500E 05
  0010001000010100
  0000010000 0001010000 1110101011 110000001  S: 0.23906500E 05
  0010001000010100
  N:  96  R:   0

**PF( 8): 0000010010

  COST: 0.24805269E 03   CPU:  37.311
  0000010010 0100010010 1011101001 001100000  S: 0.25262500E 05
  0010001000010100
  0000010010 0001010010 1110101001 110000000  S: 0.25262500E 05
  0010001000010100
  N:  64  R:   0

**PF( 9): 0000010100

  COST: 0.26157520E 03   CPU:  41.416
  0000010100 0100010100 1011101011 001100001  S: 0.36562500E 05
  0010001000010100
  0000010100 0001010100 1110101011 110000001  S: 0.36562500E 05
  0010001000010100
  N:  64  R:   0
```

```
**PF(10): 0000100000

  COST: 0.18838756E 03   CPU:  49.613
  0000100000 0000100000 1011001111 001000111  S: 0.22098500E 05
  0010001000010100
  0000100000 0000100000 1011011011 001001001  S: 0.22098500E 05
  0010001000010100
  0000100000 0000100000 1110001111 110000111  S: 0.22098500E 05
  0010001000010100
  0000100000 0000100000 1110011011 110001001  S: 0.22098500E 05
  0010001000010100
  N: 128  R:   0

**PF(11): 0000100010

  COST: 0.22909509E 03   CPU:  55.744
  0000100010 0000100010 1011001101 001000100  S: 0.23454500E 05
  0010001000010100
  0000100010 0000100010 1011011001 001001000  S: 0.23454500E 05
  0010001000010100
  0000100010 0000100010 1110001101 110000100  S: 0.23454500E 05
  0010001000010100
  0000100010 0000100010 1110011001 110001000  S: 0.23454500E 05
  0010001000010100
  N:  96  R:   0

**PF(12): 0000100100

  COST: 0.24260991E 03   CPU:  61.852
  0000100100 0000100100 1011011011 001001001  S: 0.34754500E 05
  0010001000010100
  0000100100 0000100100 1110011011 110001001  S: 0.34754500E 05
  0010001000010100
  N:  96  R:   0

**PF(13): 0000101000

  COST: 0.20901950E 03   CPU:  67.914
  0000101000 0000101000 1011010011 001000001  S: 0.27522500E 05
  0010001000010100
  0000101000 0000101000 1110010011 110000001  S: 0.27522500E 05
  0010001000010100
  N:  96  R:   0

**PF(14): 0000101010

  COST: 0.24972711E 03   CPU:  71.952
  0000101010 0000101010 1011010001 001000000  S: 0.28878500E 05
  0010001000010100
  0000101010 0000101010 1110010001 110000000  S: 0.28878500E 05
  0010001000010100
  N:  64  R:   0
```

```
**PF(15): 0000101100

  COST: 0.25640967E 03   CPU:  76.022
  0000101100 0000101100 1011010011 001000001  S: 0.40178500E 05
  0010001000010100
  0000101100 0000101100 1110010011 110000001  S: 0.40178500E 05
  0010001000010100
  N:  64  R:   0

**PF(16): 0000110000

  COST: 0.22228516E 03   CPU:  82.168
  0000110000 0000110000 1011001011 001000001  S: 0.27522500E 05
  0010001000010100
  0000110000 0000110000 1110001011 110000001  S: 0.27522500E 05
  0010001000010100
  N:  96  R:   0

**PF(17): 0000110010

  COST: 0.26299243E 03   CPU:  86.235
  0000110010 0000110010 1011001001 001000000  S: 0.28878500E 05
  0010001000010100
  0000110010 0000110010 1110001001 110000000  S: 0.28878500E 05
  0010001000010100
  N:  64  R:   0

**PF(18): 0000110100

  COST: 0.27651514E 03   CPU:  90.279
  0000110100 0000110100 1011001011 001000001  S: 0.40178500E 05
  0010001000010100
  0000110100 0000110100 1110001011 110000001  S: 0.40178500E 05
  0010001000010100
  N:  64  R:   0

**PF(19): 0001000000

  COST: 0.19188356E 03   CPU: 102.476
  0001000000 0001000000 1110101111 110000111  S: 0.20742500E 05
  0010001000010100
  0001000000 0001000000 1110111011 110001001  S: 0.20742500E 05
  0010001000010100
  N: 192  R:   0

**PF(20): 0001000010

  COST: 0.23259109E 03   CPU: 111.484
  0001000010 0001000010 1110101101 110000100  S: 0.22098500E 05
  0010001000010100
  0001000010 0001000010 1110111001 110001000  S: 0.22098500E 05
  0010001000010100
  N: 144  R:   0
```

```
**PF(21): 0001000100

  COST: 0.22953642E 03   CPU: 120.407
  0001000100 0001000110 1110111001 110001000  S: 0.28878500E 05
  0010002000010100
  N: 144  R:   0

**PF(22): 0001001000

  COST: 0.21252350E 03   CPU: 129.336
  0001001000 0001001000 1110110011 110000001  S: 0.26166500E 05
  0010001000010100
  N: 144  R:   0

**PF(23): 0001001010

  COST: 0.25323109E 03   CPU: 135.231
  0001001010 0001001010 1110110001 110000000  S: 0.27522500E 05
  0010001000010100
  N:  96  R:   0

**PF(24): 0001001100

  COST: 0.25991357E 03   CPU: 141.192
  0001001100 0001001100 1110110011 110000001  S: 0.38822500E 05
  0010001000010100
  N:  96  R:   0

**PF(25): 0001010000

  COST: 0.22578911E 03   CPU: 150.184
  0001010000 0001010000 1110101011 110000001  S: 0.26166500E 05
  0010001000010100
  N: 144  R:   0

**PF(26): 0001010010

  COST: 0.26649634E 03   CPU: 156.101
  0001010010 0001010010 1110101001 110000000  S: 0.27522500E 05
  0010001000010100
  N:  96  R:   0

**PF(27): 0001010100

  COST: 0.27023193E 03   CPU: 161.993
  0001010100 0001010110 1110101001 110000000  S: 0.34302500E 05
  0010002000010100
  N:  96  R:   0
```

```
**PF(28): 0010000000

  COST: 0.18838756E 03   CPU: 168.150
  0010000000 0010000000 1001101111 000100111  S: 0.22098500E 05
  0010001000010100
  0010000000 0010000000 1001111011 000101001  S: 0.22098500E 05
  0010001000010100
  N:  96  R:   0

**PF(29): 0010000010

  COST: 0.22909509E 03   CPU: 173.214
  0010000010 0010000010 1001101101 000100100  S: 0.23454500E 05
  0010001000010100
  0010000010 0010000010 1001111001 000101000  S: 0.23454500E 05
  0010001000010100
  N:  80  R:   0

**PF(30): 0010000100

  COST: 0.23378389E 03   CPU: 178.211
  0010000100 0011000100 1100111011 100001001  S: 0.32494500E 05
  0010001000010100
  N:  80  R:   0

**PF(31): 0010001000

  COST: 0.20018150E 03   CPU: 184.190
  0010001000 0010001000 1100110011 100000001  S: 0.23454500E 05
  0010001000010100
  N:  96  R:   0

**PF(32): 0010001010

  COST: 0.24088911E 03   CPU: 188.149
  0010001010 0010001010 1100110001 100000000  S: 0.24810500E 05
  0010001000010100
  N:  64  R:   0

**PF(33): 0010001100

  COST: 0.24757190E 03   CPU: 192.159
  0010001100 0010001100 1100110011 100000001  S: 0.36110500E 05
  0010001000010100
  N:  64  R:   0

**PF(34): 0010010000

  COST: 0.21345917E 03   CPU: 198.159
  0010010000 0011010000 1100101011 100000001  S: 0.25262500E 05
  0010001000010100
  N:  96  R:   0
```

```
**PF(35): 0010010010

  COST: 0.25416669E 03   CPU: 202.115
  0010010010 0011010010 1100101001 100000000  S: 0.26618500E 05
  0010001000010100
  N:  64  R:   0

**PF(36): 0010010100

  COST: 0.26768921E 03   CPU: 206.067
  0010010100 0011010100 1100101011 100000001  S: 0.37918500E 05
  0010001000010100
  N:  64  R:   0

**PF(37): 0010100000

  COST: 0.19450156E 03   CPU: 212.098
  0010100000 0010100000 1100001111 100000111  S: 0.23454500E 05
  0010001000010100
  0010100000 0010100000 1100011011 100001001  S: 0.23454500E 05
  0010001000010100
  N:  96  R:   0

**PF(38): 0010100010

  COST: 0.23520912E 03   CPU: 216.600
  0010100010 0010100010 1100001101 100000100  S: 0.24810500E 05
  0010001000010100
  0010100010 0010100010 1100011001 100001000  S: 0.24810500E 05
  0010001000010100
  N:  72  R:   0

**PF(39): 0010100100

  COST: 0.24872389E 03   CPU: 221.079
  0010100100 0010100100 1100011011 100001001  S: 0.36110500E 05
  0010001000010100
  N:  72  R:   0

**PF(40): 0010101000

  COST: 0.21513350E 03   CPU: 225.537
  0010101000 0010101000 1100010011 100000001  S: 0.28878500E 05
  0010001000010100
  N:  72  R:   0

**PF(41): 0010101010

  COST: 0.25584109E 03   CPU: 228.502
  0010101010 0010101010 1100010001 100000000  S: 0.30234500E 05
  0010001000010100
  N:  48  R:   0
```

```
**PF(42): 0010101100

  COST: 0.26252344E 03   CPU: 231.499
  0010101100 0010101100 1100010011 100000001  S: 0.41534500E 05
  0010001000010100
  N:  48  R:   0

**PF(43): 0010110000

  COST: 0.22839911E 03   CPU: 236.030
  0010110000 0010110000 1100001011 100000001  S: 0.28878500E 05
  0010001000010100
  N:  72  R:   0

**PF(44): 0010110010

  COST: 0.26910645E 03   CPU: 239.021
  0010110010 0010110010 1100001001 100000000  S: 0.30234500E 05
  0010001000010100
  N:  48  R:   0

**PF(45): 0010110100

  COST: 0.28262915E 03   CPU: 242.006
  0010110100 0010110100 1100001011 100000001  S: 0.41534500E 05
  0010001000010100
  N:  48  R:   0

**PF(46): 0011000000

  COST: 0.19599956E 03   CPU: 250.154
  0011000000 0011000000 1100101111 100000111  S: 0.22098500E 05
  0010001000010100
  0011000000 0011000000 1100111011 100001001  S: 0.22098500E 05
  0010001000010100
  N: 128  R:   0

**PF(47): 0011000010

  COST: 0.23670711E 03   CPU: 256.192
  0011000010 0011000010 1100101101 100000100  S: 0.23454500E 05
  0010001000010100
  0011000010 0011000010 1100111001 100001000  S: 0.23454500E 05
  0010001000010100
  N:  96  R:   0

**PF(48): 0011000100

  COST: 0.25022[illegible]89E 03   CPU: 262.159
  0011000100 0[illegible]11000100 1100111011 100001001  S: 0.34754500E 05
  00100010000[illegible]100
  N:  96  R:   0
```

```
**PF(49): 0011001000

  COST: 0.21663951E 03   CPU: 268.124
  0011001000 0011001000 1100110011 100000001  S: 0.27522500E 05
  0010001000010100
  N:  96  R:   0

**PF(50): 0011001010

  COST: 0.25734668E 03   CPU: 272.073
  0011001010 0011001010 1100110001 100000000  S: 0.28878500E 05
  0010001000010100
  N:  64  R:   0

**PF(51): 0011001100

  COST: 0.26402954E 03   CPU: 276.090
  0011001100 0011001100 1100110011 100000001  S: 0.40178500E 05
  0010001000010100
  N:  64  R:   0

**PF(52): 0011010000

  COST: 0.22990511E 03   CPU: 282.135
  0011010000 0011010000 1100101011 100000001  S: 0.27522500E 05
  0010001000010100
  N:  96  R:   0

**PF(53): 0011010010

  COST: 0.27061230E 03   CPU: 286.106
  0011010010 0011010010 1100101001 100000000  S: 0.28878500E 05
  0010001000010100
  N:  64  R:   0

**PF(54): 0011010100

  COST: 0.28413525E 03   CPU: 290.089
  0011010100 0011010100 1100101011 100000001  S: 0.40178500E 05
  0010001000010100
  N:  64  R:   0

**PF(55): 0100000000

  COST: 0.19188356E 03   CPU: 300.349
  0100000000 0100000000 1011101111 001100111  S: 0.20742500E 05
  0010001000010100
  0100000000 0100000000 1011111011 001101001  S: 0.20742500E 05
  0010001000010100
  N: 160  R:   0
```

```
**PF(56): 0100000010

  COST: 0.23259109E 03   CPU: 308.444
  0100000010 0100000010 1011101101 001100100  S: 0.22098500E 05
  0010001000010100
  0100000010 0100000010 1011111001 001101000  S: 0.22098500E 05
  0010001000010100
  N: 128  R:   0

**PF(57): 0100000100

  COST: 0.22953642E 03   CPU: 316.396
  0100000100 0100000110 1011111001 001101000  S: 0.28878500E 05
  0010002000010100
  N: 128  R:   0

**PF(58): 0100001000

  COST: 0.20333751E 03   CPU: 325.395
  0100001000 0100001000 1010110011 000000001  S: 0.23454500E 05
  0010001000010100
  N: 144  R:   0

**PF(59): 0100001010

  COST: 0.24404510E 03   CPU: 331.347
  0100001010 0100001010 1010' )001 000000000  S: 0.24810500E 05
  0010001000010100
  N:  96  R:   0

**PF(60): 0100001100

  COST: 0.25072789E 03   CPU: 337.344
  0100001100 0100001100 1010110011 000000001  S: 0.36110500E 05
  0010001000010100
  N:  96  R:   0

**PF(61): 0100010000

  COST: 0.21661516E 03   CPU: 346.152
  0100010000 0101010000 1010101011 000000001  S: 0.25262500E 05
  0010001000010100
  N: 144  R:   0

**PF(62): 0100010010

  COST: 0.25732227E 03   CPU: 351.952
  0100010010 0101010010 1010101001 000000000  S: 0.26618500E 05
  0010001000010100
  N:  96  R:   0
```

```
**PF(63): 0100010100

  COST: 0.27023193E 03   CPU: 357.899
  0100010100 0100010110 1011101001 001100000  S: 0.34302500E 05
  0010002000010100
  N:  96  R:   0

**PF(64): 0100100000

  COST: 0.19765756E 03   CPU: 366.142
  0100100000 0100100000 1010001111 000000111  S: 0.23454500E 05
  0010001000010100
  0100100000 0100100000 1010011011 000001001  S: 0.23454500E 05
  0010001000010100
  N: 128  R:   0

**PF(65): 0100100010

  COST: 0.23836511E 03   CPU: 372.274
  0100100010 0100100010 1010001101 000000100  S: 0.24810500E 05
  0010001000010100
  0100100010 0100100010 1010011001 000001000  S: 0.24810500E 05
  0010001000010100
  N:  96  R:   0

**PF(66): 0100100100

  COST: 0.25187988E 03   CPU: 378.349
  0100100100 0100100100 10100 [illegible]011 000001001  S: 0.36110500E 05
  0010001000010100
  N:  96  R:   0

**PF(67): 0100101000

  COST: 0.21828951E 03   CPU: 384.432
  0100101000 0100101000 1010010011 000000001  S: 0.28878500E 05
  0010001000010100
  N:  96  P:   0

**PF(68): 0100101010

  COST: 0.25899683E 03   CPU: 388.453
  0100101010 0100101010 1010010001 000000000  S: 0.30234500E 05
  0010001000010100
  N:  64  R:   0

**PF(69): 0100101100

  COST: 0.26567969E 03   CPU: 392.515
  0100101100 0100101100 1010010011 000000001  S: 0.41534500E 05
  0010001000010100
  N:  64  P:   0
```

```
**PF(70): 0100110000

  COST: 0.23155510E 03   CPU: 398.665
  0100110000 0100110000 1010001011 000000001  S: 0.28878500E 05
  0010001000010100
  N:  96  R:   0

**PF(71): 0100110010

  COST: 0.27226245E 03   CPU: 402.717
  0100110010 0100110010 1010001001 000000000  S: 0.30234500E 05
  0010001000010100
  N•  64  R:   0

**PF(72): 0100110100

  COST: 0.28578516E 03   CPU: 406.750
  0100110100 0100110100 1010001011 000000001  S: 0.41534500E 05
  0010001000010100
  N:  64  R:   0

**PF(73): 0101000000

  COST: 0.20115356E 03   CPU: 414.999
  0101000000 0101000000 1010101111 000000111  S: 0.22098500E 05
  0010001000010100
  0101000000 0101000000 1010111011 000001001  S: 0.22098500E 05
  0010001000010100
  N: 128  R:   0

**PF(74): 0101000010

  COST: 0.24186110E 03   CPU: 421.120
  0101000010 0101000010 1010101101 000000100  S: 0.23454500E 05
  0010001000010100
  0101000010 0101000010 1010111001 000001000  S: 0.23454500E 05
  0010001000010100
  N:  96  R:   0

**PF(75): 0101000100

  COST: 0.25538390E 03   CPU: 427.200
  0101000100 0101000100 1010111011 000001001  S: 0.34754500E 05
  0010001000010100
  N:  96  R:   0

**PF(76): 0101001000

  COST: 0.22179350E 03   CPU: 433.278
  0101001000 0101001000 1010110011 000000001  S: 0.27522500E 05
  0010001000010100
  N:  96  R:   0
```

```
**PF(77): 0101001010

  COST: 0.26250049E 03   CPU: 437.294
  0101001010 0101001010 1010110001 000000000  S: 0.28878500E 05
  0010001000010100
  N:  64  R:   0

**PF(78): 0101001100

  COST: 0.26918359E 03   CPU: 441.349
  0101001100 0101001100 1010110011 000000001  S: 0.40178500E 05
  0010001000010100
  N:  64  R:   0

**PF(79): 0101010000

  COST: 0.23505910E 03   CPU: 447.451
  0101010000 0101010000 1010101011 000000001  S: 0.27522500E 05
  0010001000010100
  N:  96  R:   0

**PF(80): 0101010010

  COST: 0.27576636E 03   CPU: 451.474
  0101010010 0101010010 1010101001 000000000  S: 0.28878500E 05
  0010001000010100
  N:  64  R:   0

**PF(81): 0101010100

  COST: 0.28928882E 03   CPU: 455.490
  0101010100 0101010100 1010101011 000000001  S: 0.40178500E 05
  0010001000010100
  N:  64  R:   0

PF(28) CONTAINS BEST SOLUTION(S).   7232 SOLUTIONS CONSIDERED.

   MEAN COST OVER ALL SOLUTIONS:   0.28151123E 03
   MAXIMUM COST ENCOUNTERED:       0.42425903E 03

********** SOLUTION TERMINATED **********
```

# Appendix G

## SOLUTION LISTING FOR CASE 3

## OF THE MEDICAL DIAGNOSIS PROBLEM

```
********** SOLUTION BEGINS **********

**PF( 1): 0000000000

  COST: 0.27245435E 03   CPU:   4.065
  0000000000 0001010000 1110101111 110000111  S: 0.30686500E 05
  0010001000010100
  N:  64  R:   0

**PF( 2): 0000000010

  COST: 0.31508911E 03   CPU:   8.025
  0000000010 0001000010 1110101100 110000100  S: 0.15544500E 05
  0010001000010100
  N:  64  R:   0

**PF( 3): 0000000100

  COST: 0.29853931E 03   CPU:  11.904
  0000000100 0100000100 1011111010 001101000  S: 0.32042500E 05
  0010001000010100
  0000000100 0001000100 1110111010 110001000  S: 0.32042500E 05
  0010001000010100
  N:  64  R:   0

**PF( 4): 0000001000

  COST: 0.26675464E 03   CPU:  17.696
  0000001000 0000001100 1011110011 001100001  S: 0.33398500E 05
  0010001000010100
  0000001000 0000001100 1110110011 110000001  S: 0.33398500E 05
  0010001000010100
  N:  96  R:   0
```

```
**PF( 5): 0000001010

  COST: 0.32318921E 03   CPU:  21.550
  0000001010 0000001010 1011110000 001100000  S: 0.18256500E 05
  0010001000010100
  0000001010 0000001010 1110110000 110000000  S: 0.18256500E 05
  0010001000010100
  N:  64  R:   0

**PF( 6): 0000001100

  COST: 0.31553931E 03   CPU:  25.466
  0000001100 0000001100 1011110010 001100000  S: 0.34754500E 05
  0010001000010100
  0000001100 0000001100 1110110010 110000000  S: 0.34754500E 05
  0010001000010100
  N:  64  R:   0

**PF( 7): 0000010000

  COST: 0.27250854E 03   CPU:  31.468
  0000010000 0100010100 1011101011 001100001  S: 0.34754500E 05
  0010001000010100
  0000010000 0001010100 1110101011 110000001  S: 0.34754500E 05
  0010001000010100
  N:  96  R:   0

**PF( 8): 0000010010

  COST: 0.32894312E 03   CPU:  35.421
  0000010010 0100010010 1011101000 001100000  S: 0.19612500E 05
  0010001000010100
  0000010010 0001010010 1110101000 110000000  S: 0.19612500E 05
  0010001000010100
  N:  64  R:   0

**PF( 9): 0000010100

  COST: 0.32619287E 03   CPU:  39.374
  0000010100 0100010100 1011101010 001100000  S: 0.36110500E 05
  0010001000010100
  0000010100 0001010100 1110101010 110000000  S: 0.36110500E 05
  0010001000010100
  N:  64  R:   0
```

```
**PF(10): 0000100000

  COST: 0.26295435E 03   CPU:  47.330
  0000100000 0000110000 1011001111 001000111  S: 0.33398500E 05
  0010001000010100
  0000100000 0000100100 1011011011 001001001  S: 0.33398500E 05
  0010001000010100
  0000100000 0000110000 1110001111 110000111  S: 0.33398500E 05
  0010001000010100
  0000100000 0000100100 1110011011 110001001  S: 0.33398500E 05
  0010001000010100
  N: 128  R:   0

**PF(11): 0000100010

  COST: 0.31938916E 03   CPU:  53.276
  0000100010 0000100010 1011001100 001000100  S: 0.18256500E 05
  0010001000010100
  0000100010 0000100010 1011011000 001001000  S: 0.18256500E 05
  0010001000010100
  0000100010 0000100010 1110001100 110000100  S: 0.18256500E 05
  0010001000010100
  0000100010 0000100010 1110011000 110001000  S: 0.18256500E 05
  0010001000010100
  N:  96  R:   0

**PF(12): 0000100100

  COST: 0.31633936E 03   CPU:  59.187
  0000100100 0000100100 1011011010 001001000  S: 0.34754500E 05
  0010001000010100
  0000100100 0000100100 1110011010 110001000  S: 0.34754500E 05
  0010001000010100
  N:  96  R:   0

**PF(13): 0000101000

  COST: 0.28635449E 03   CPU:  65.039
  0000101000 0000101100 1011010011 001000001  S: 0.37466500E 05
  0010001000010100
  0000101000 0000101100 1110010011 110000001  S: 0.37466500E 05
  0010001000010100
  N:  96  P:   0

**PF(14): 0000'01010

  COST: 0.34278906E 03   CPU:  68.942
  0000101010 0000101010 1011010000 001000000  S: 0.22324500E 05
  0010001000010100
  0000101010 0000101010 1110010000 110000000  S: 0.22324500E 05
  0010001000010100
  N:  64  P:   0
```

```
**PF(15): 0000101100

  COST: 0.33513940E 03   CPU:  72.880
  0000101100 0000101100 1011010010 001000000  S: 0.38822500E 05
  0010001000010100
  0000101100 0000101100 1110010010 110000000  S: 0.38822500E 05
  0010001000010100
  N:  64  R:   0

**PF(16): 0000110000

  COST: 0.29030835E 03   CPU:  78.796
  0000110000 0000110100 1011001011 001000001  S: 0.37466500E 05
  0010001000010100
  0000110000 0000110100 1110001011 110000001  S: 0.37466500E 05
  0010001000010100
  N:  96  R:   0

**PF(17): 0000110010

  COST: 0.34674292E 03   CPU:  82.705
  0000110010 0000110010 1011001000 001000000  S: 0.22324500E 05
  0010001000010100
  0000110010 0000110010 1110001000 110000000  S: 0.22324500E 05
  0010001000010100
  N:  64  R:   0

**PF(18): 0000110100

  COST: 0.34399292E 03   CPU:  86.640
  0000110100 0000110100 1011001010 001000000  S: 0.38822500E 05
  0010001000010100
  0000110100 0000110100 1110001010 110000000  S: 0.38822500E 05
  0010001000010100
  N:  64  R:   0

**PF(19): 0001000000

  COST: 0.25885449E 03   CPU:  98.333
  0001000000 0001010000 1110101111 110000111  S: 0.32494500E 05
  0010001000010100
  0001000000 0001000100 1110111011 110001001  S: 0.32494500E 05
  0010001000010100
  N: 192  R:   0

**PF(20): 0001000010

  COST: 0.31528906E 03   CPU: 106.930
  0001000010 0001000010 1110101100 110000100  S: 0.17352500E 05
  0010001000010100
  0001000010 0001000010 1110111000 110001000  S: 0.17352500E 05
  0010001000010100
  N: 144  R:   0
```

```
**PF(21): 0001000100

  COST: 0.31253931E 03   CPU: 115.413
  0001000100 0001000100 1110111010 110001000  S: 0.33850500E 05
  0010001000010100
  N: 144  R:   0

**PF(22): 0001001000

  COST: 0.28255469E 03   CPU: 123.930
  0001001000 0001001100 1110110011 110000001  S: 0.36562500E 05
  0010001000010100
  N: 144  R:   0

**PF(23): 0001001010

  COST: 0.33898901E 03   CPU: 129.528
  0001001010 0001001010 1110110000 110000000  S: 0.21420500E 05
  0010001000010100
  N:  96  R:   0

**PF(24): 0001001100

  COST: 0.33133936E 03   CPU: 135.240
  0001001100 0001001100 1110110010 110000000  S: 0.37918500E 05
  0010001000010100
  N:  96  R:   0

**PF(25): 0001010000

  COST: 0.28650854E 03   CPU: 143.812
  0001010000 0001010100 1110101011 110000001  S: 0.36562500E 05
  0010001000010100
  N: 144  R:   0

**PF(26): 0001010010

  COST: 0.34294312E 03   CPU: 149.388
  0001010010 0001010010 1110101000 110000000  S: 0.21420500E 05
  0010001000010100
  N:  96  R:   0

**PF(27): 0001010100

  COST: 0.34019287E 03   CPU: 155.100
  0001010100 0001010100 1110101010 110000000  S: 0.37918500E 05
  0010001000010100
  N:  96  R:   0
```

```
**PF(28): 0010000000

   COST: 0.26295435E 03   CPU: 160.957
   0010000000 0010010000 1001101111 000100111  S: 0.33398500E 05
   0010001000010100
   0010000000 0010000100 1001111011 000101001  S: 0.33398500E 05
   0010001000010100
   N:  96  R:   0

**PF(29): 0010000010

   COST: 0.31938916E 03   CPU: 165.814
   0010000010 0010000010 1001101100 000100100  S: 0.18256500E 05
   0010001000010100
   0010000010 0010000010 1001111000 000101000  S: 0.18256500E 05
   0010001000010100
   N:  80  R:   0

**PF(30): 0010000100

   COST: 0.31323926E 03   CPU: 170.606
   0010000100 0011000100 1100111010 100001000  S: 0.33398500E 05
   0010001000010100
   N:  80  R:   0

**PF(31): 0010001000

   COST: 0.28145459E 03   CPU: 176.327
   0010001000 0010001100 1100110011 100000001  S: 0.34754500E 05
   0010001000010100
   N:  96  R:   0

**PF(32): 0010001010

   COST: 0.33788892E 03   CPU: 180.070
   0010001010 0010001010 1100110000 100000000  S: 0.19612500E 05
   0010001000010100
   N:  64  R:   0

**PF(33): 0010001100

   COST: 0.33023926E 03   CPU: 183.949
   0010001100 0010001100 1100110010 100000000  S: 0.36110500E 05
   0010001000010100
   N:  64  R:   0

**PF(34): 0010010000

   COST: 0.28720850E 03   CPU: 189.739
   0010010000 0011010100 1100101011 100000001  S: 0.36110500E 05
   0010001000010100
   N:  96  R:   0
```

```
**PF(35): 0010010010

   COST: 0.34364307E 03   CPU: 193.514
   0010010010 0011010010 1100101000 100000000  S: 0.20968500E 05
   0010001000010100
   N:  64  R:   0

**PF(36): 0010010100

   COST: 0.34089307E 03   CPU: 197.332
   0010010100 0011010100 1100101010 100000000  S: 0.37466500E 05
   0010001000010100
   N:  64  R:   0

**PF(37): 0010100000

   COST: 0.27765430E 03   CPU: 203.134
   0010100000 0010110000 1100001111 100000111  S: 0.34754500E 05
   0010001000010100
   0010100000 0010100100 1100011011 100001001  S: 0.34754500E 05
   0010001000010100
   N:  96  R:   0

**PF(38): 0010100010

   COST: 0.33408911E 03   CPU: 207.387
   0010100010 0010100010 1100001100 100000100  S: 0.19612500E 05
   0010001000010100
   0010100010 0010100010 1100011000 100001000  S: 0.19612500E 05
   0010001000010100
   N:  72  R:   0

**PF(39): 0010100100

   COST: 0.33103931E 03   CPU: 211.680
   0010100100 0010100100 1100011010 100001000  S: 0.36110500E 05
   0010001000010100
   N:  72  R:   0

**PF(40): 0010101000

   COST: 0.30105444E 03   CPU: 215.992
   0010101000 0010101100 1100010011 100000001  S: 0.38822500E 05
   0010001000010100
   N:  72  R:   0

**PF(41): 0010101010

   COST: 0.35748901E 03   CPU: 218.807
   0010101010 0010101010 1100010000 100000000  S: 0.23680500E 05
   0010001000010100
   N:  48  R:   0
```

```
**PF(42): 0010101100

  COST: 0.34983911E 03   CPU: 221.700
  0010101100 0010101100 1100010010 100000000  S: 0.40178500E 05
  0010001000010100
  N:  48  R:    0

**PF(43): 0010110000

  COST: 0.30500830E 03   CPU: 226.028
  0010110000 0010110100 1100001011 100000001  S: 0.38822500E 05
  0010001000010100
  N:  72  R:    0

**PF(44): 0010110010

  COST: 0.36144312E 03   CPU: 228.923
  0010110010 0010110010 1100001000 100000000  S: 0.23680500E 05
  0010001000010100
  N:  48  R:    0

**PF(45): 0010110100

  COST: 0.35869287E 03   CPU: 231.157
  0010110100 0010110100 1100001010 100000000  S: 0.40178500E 05
  0010001000010100
  N:  48  R:    0

**PF(46): 0011000000

  COST: 0.27165430E 03   CPU: 239.698
  0011000000 0011010000 1100101111 100000111  S: 0.33850500E 05
  0010001000010100
  0011000000 0011000100 1100111011 100001001  S: 0.33850500E 05
  0010001000010100
  N: 128  R:    0

**PF(47): 0011000010

  COST: 0.32808911E 03   CPU: 245.550
  0011000010 0011000010 1100101100 100000100  S: 0.18708500E 05
  0010001000010100
  0011000010 0011000010 1100111000 100001000  S: 0.18708500E 05
  0010001000010100
  N:  96  R:    0

**PF(48): 0011000100

  COST: 0.32533911E 03   CPU: 251.314
  0011000100 0011000100 1100111010 100001000  S: 0.35206500E 05
  0010001000010100
  N:  96  R:    0
```

```
**PF(49): 0011001000

  COST: 0.29535449E 03   CPU: 257.043
  0011001000 0011001100 1100110011 100000001  S: 0.37918500E 05
  001000100001010O
  N:  96  R:   0

**PF(50): 0011001010

  COST: 0.35178906E 03   CPU: 260.826
  0011001010 0011001010 1100110000 100000000  S: 0.22776500E 05
  001000100001010O
  N:  64  R:   0

**PF(51): 0011001100

  COST: 0.34413916E 03   CPU: 264.714
  0011001100 0011001100 1100110010 100000000  S: 0.39274500E 05
  001000100001010O
  N:  64  R:   0

**PF(52): 0011010000

  COST: 0.29930835E 03   CPU: 270.519
  0011010000 0011010100 1100101011 100000001  S: 0.37918500E 05
  001000100001010O
  N:  96  R:   0

**PF(53): 0011010010

  COST: 0.35574316E 03   CPU: 274.326
  0011010010 0011010010 1100101000 100000000  S: 0.22776500E 05
  001000100001010O
  N:  64  R:   0

**PF(54): 0011010100

  COST: 0.35299292E 03   CPU: 278.133
  0011010100 0011010100 1100101010 100000000  S: 0.39274500E 05
  001000100001010O
  N:  64  R:   0

**PF(55): 0100000000

  COST: 0.25885449E 03   CPU: 287.964
  0100000000 0100010000 1011101111 001100111  S: 0.32494500E 05
  001000100001010O
  0100000000 0100000100 1011111011 001101001  S: 0.32494500E 05
  001000100001010O
  N: 160  R:   0
```

```
**PF(56): 0100000010

  COST: 0.31528906E 03   CPU: 295.706
  0100000010 0100000010 1011101100 001100100  S: 0.17352500E 05
  0010001000010100
  0100000010 0100000010 1011111000 001101000  S: 0.17352500E 05
  0010001000010100
  N: 128  R:   0

**PF(57): 0100000100

  COST: 0.31083911E 03   CPU: 303.393
  0100000100 0101000100 1010111010 000001000  S: 0.33398500E 05
  0010001000010100
  N: 128  R:   0

**PF(58): 0100001000

  COST: 0.27905444E 03   CPU: 311.970
  0100001000 0100001100 1010110011 000000001  S: 0.34754500E 05
  0010001000010100
  N: 144  R:   0

**PF(59): 0100001010

  COST: 0.33548901E 03   CPU: 317.647
  0100001010 0100001010 1010110000 000000000  S: 0.19612500E 05
  0010001000010100
  N:  96  R:   0

**PF(60): 0100001100

  COST: 0.32783911E 03   CPU: 323.387
  0100001100 0100001100 1010110010 000000000  S: 0.35110500E 05
  0010001000010100
  N:  96  R:   0

**PF(61): 0100010000

  COST: 0.28480830E 03   CPU: 332.016
  0100010000 0101010100 1010101011 000000001  S: 0.36110500E 05
  0010001000010100
  N: 144  R:   0

**PF(62): 0100010010

  COST: 0.34124292E 03   CPU: 337.703
  0100010010 0101010010 1010101000 000000000  S: 0.20968500E 05
  0010001000010100
  N:  96  R:   0
```

```
**PF(63): 0100010100

  COST: 0.33849316E 03   CPU: 343.408
  0100010100 0101010100 1010101010 000000000  S: 0.37466500E 05
  0010001000010100
  N:  96  R:   0

**PF(64): 0100100000

  COST: 0.27525439E 03   CPU: 351.120
  0100100000 0100110000 1010001111 000000111  S: 0.34754500E 05
  0010001000010100
  0100100000 0100100100 1010011011 000001001  S: 0.34754500E 05
  0010001000010100
  N: 128  R:   0

**PF(65): 0100100010

  COST: 0.33168921E 03   CPU: 356.826
  0100100010 0100100010 1010001100 000000100  S: 0.19612500E 05
  0010001000010100
  0100100010 0100100010 1010011000 000001000  S: 0.19612500E 05
  0010001000010100
  N:  96  R:   0

**PF(66): 0100100100

  COST: 0.32863916E 03   CPU: 362.515
  0100100100 0100100100 1010011010 000001000  S: 0.36110500E 05
  0010001000010100
  N:  96  R:   0

**PF(67): 0100101000

  COST: 0.29865454E 03   CPU: 368.200
  0100101000 0100101100 1010010011 000000001  S: 0.38822500E 05
  0010001000010100
  N:  96  R:   0

**PF(68): 0100101010

  COST: 0.35508911E 03   CPU: 371.955
  0100101010 0100101010 1010010000 000000000  S: 0.23680500E 05
  0010001000010100
  N:  64  R:   0

**PF(69): 0100101100

  COST: 0.34743896E 03   CPU: 375.765
  0100101100 0100101100 1010010010 000000000  S: 0.40178500E 05
  0010001000010100
  N:  64  R:   0
```

```
**PF(70): 0100110000

  COST: 0.30260840E 03   CPU: 381.511
  0100110000 0100110100 1010001011 000000001  S: 0.38822500E 05
  0010001000010100
  N:  96  R:   0

**PF(71): 0100110010

  COST: 0.35904321E 03   CPU: 385.298
  0100110010 0100110010 1010001000 000000000  S: 0.23680500E 05
  0010001000010100
  N:  64  R:   0

**PF(72): 0100110100

  COST: 0.35629297E 03   CPU: 389.080
  0100110100 0100110100 1010001010 000000000  S: 0.40178500E 05
  0010001000010100
  N:  64  R:   0

**PF(73): 0101000000

  COST: 0.27115430E 03   CPU: 396.785
  0101000000 0101010000 1010101111 000000111  S: 0.33850500E 05
  0010001000010100
  0101000000 0101000100 1010111011 000001001  S: 0.33850500E 05
  0010001000010100
  N: 128  R:   0

**PF(74): 0101000010

  COST: 0.32758911E 03   CPU: 402.623
  0101000010 0101000010 1010101100 000000100  S: 0.18708500E 05
  0010001000010100
  0101000010 0101000010 1010111000 000001000  S: 0.18708500E 05
  0010001000010100
  N:  96  R:   0

**PF(75): 0101000100

  COST: 0.32483911E 03   CPU: 408.315
  0101000100 0101000100 1010111010 000001000  S: 0.35206500E 05
  0010001000010100
  N:  96  R:   0

**PF(76): 0101001000

  COST: 0.29485449E 03   CPU: 413.986
  0101001000 0101001100 1010110011 000000001  S: 0.37918500E 05
  0010001000010100
  N:  96  R:   0
```

```
**PF(77): 0101001010

   COST: 0.35128906E 03    CPU: 417.735
   0101001010 0101001010 1010110000 000000000  S: 0.22776500E 05
   0010001000010100
   N:  64  R:   0

**PF(78): 0101001100

   COST: 0.34363892E 03    CPU: 421.520
   0101001100 0101001100 1010110010 000000000  S: 0.39274500E 05
   0010001000010100
   N:  64  R:   0

**PF(79): 0101010000

   COST: 0.29880811E 03    CPU: 427.219
   0101010000 0101010100 1010101011 000000001  S: 0.37918500E 05
   0010001000010100
   N:  96  R:   0

**PF(80): 0101010010

   COST: 0.35524292E 03    CPU: 430.977
   0101010010 0101010010 1010101000 000000000  S: 0.22776500E 05
   0010001000010100
   N:  64  R:   0

**PF(81): 0101010100

   COST: 0.35249316E 03    CPU: 434.735
   0101010100 0101010100 1010101010 000000000  S: 0.39274500E 05
   0010001000010100
   N:  64  R:   0

PF(55) CONTAINS BEST SOLUTION(S).   7232 SOLUTIONS CONSIDERED.

    MEAN COST OVER ALL SOLUTIONS:    0.35939697E 03
    MAXIMUM COST ENCOUNTERED:        0.58598242E 03

********** SOLUTION TERMINATED **********
```

# Appendix H

## SOLUTION LISTING FOR PRIMITIVE OPERATION $Q_7$ IN THE MEDICAL DIAGNOSIS PROBLEM

```
********** SOLUTION BEGINS **********

**PF( 1): 0000000000

  COST: 0.31459985E 03   CPU:   2.052
  0000000000 0100000000 1011111111 001101111  S: 0.19160500E 05
  0000001000000000
  0000000000 0001000000 1110111111 110001111  S: 0.19160500E 05
  0000001000000000
  N:  64  R:   0

**PF( 2): 0000000010

  COST: 0.38659985E 03   CPU:   4.135
  0000000010 0100000010 1011111101 001101100  S: 0.19520500E 05
  0000001000000000
  0000000010 0001000010 1110111101 110001100  S: 0.19520500E 05
  0000001000000000
  N:  64  R:   0

**PF( 3): 0000000100

  COST: 0.37699976E 03   CPU:   6.150
  0000000100 0100000100 1011111011 001101001  S: 0.20360500E 05
  0000001000000000
  0000000100 0001000100 1110111011 110001001  S: 0.20360500E 05
  0000001000000000
  N:  64  R:   0

**PF( 4): 0000001000

  COST: 0.32059985E 03   CPU:   9.037
  0000001000 0100001000 1011110011 001100001  S: 0.15560500E 05
  0000001000000000
  0000001000 0001001000 1110110011 110000001  S: 0.15560500E 05
  0000001000000000
  N:  96  R:   0
```

```
**PF( 5): 0000001010

  COST: 0.39019995E 03   CPU:  11.008
  0000001010 0100001010 1011110001 001100000  S: 0.15920500E 05
  0000001000000000
  0000001010 0001001010 1110110001 110000000  S: 0.15920500E 05
  0000001000000000
  N:  64  R:   0

**PF( 6): 0000001100

  COST: 0.40099976E 03   CPU:  13.001
  0000001100 0100001100 1011110011 001100001  S: 0.23960500E 05
  0000001000000000
  0000001100 0001001100 1110110011 110000001  S: 0.23960500E 05
  0000001000000000
  N:  64  R:   0

**PF( 7): 0000010000

  COST: 0.34219971E 03   CPU:  15.927
  0000010000 0100010000 1011101011 001100001  S: 0.15560500E 05
  0000001000000000
  0000010000 0001010000 1110101011 110000001  S: 0.15560500E 05
  0000001000000000
  N:  96  R:   0

**PF( 8): 0000010010

  COST: 0.41179980E 03   CPU:  17.921
  0000010010 0100010010 1011101001 001100000  S: 0.15920500E 05
  0000001000000000
  0000010010 0001010010 1110101001 110000000  S: 0.15920500E 05
  0000001000000000
  N:  64  R:   0

**PF( 9): 0000010100

  COST: 0.43459985E 03   CPU:  19.859
  0000010100 0100010100 1011100011 001100001  S: 0.23960500E 05
  0000001000000000
  0000010100 0100010100 1011101011 001100001  S: 0.23960500E 05
  0000001000000000
  0000010100 0001010100 1110100011 110000001  S: 0.23960500E 05
  0000001000000000
  0000010100 0001010100 1110101011 110000001  S: 0.23960500E 05
  0000001000000000
  N:  64  R:   0
```

```
**PF(10): 0000100000

  COST: 0.30859985E 03   CPU:  23.736
  0000100000 0100100000 1011001111 001000111  S: 0.15560500E 05
  0000001000000000
  0000100000 0100100000 1011011011 001001001  S: 0.15560500E 05
  0000001000000000
  0000100000 0001100000 1110001111 110000111  S: 0.15560500E 05
  0000001000000000
  0000100000 0001100000 1110011011 110001001  S: 0.15560500E 05
  0000001000000000
  N: 128  P:   0

**PF(11): 0000100010

  COST: 0.37819971E 03   CPU:  26.644
  0000100010 0100100010 1011001101 001000100  S: 0.15920500E 05
  0000001000000000
  0000100010 0100100010 1011011001 001001000  S: 0.15920500E 05
  0000001000000000
  0000100010 0001100010 1110001101 110000100  S: 0.15920500E 05
  0000001000000000
  0000100010 0001100010 1110011001 110001000  S: 0.15920500E 05
  0000001000000000
  N:  96  P:   0

**PF(12): 0000100100

  COST: 0.40099976E 03   CPU:  29.525
  0000100100 0100100100 1011010011 001000001  S: 0.23960500E 05
  0000001000000000
  0000100100 0100100100 1011011011 001001001  S: 0.23960500E 05
  0000001000000000
  0000100100 0001100100 1110010011 110000001  S: 0.23960500E 05
  0000001000000000
  0000100100 0001100100 1110011011 110001001  S: 0.23960500E 05
  0000001000000000
  N:  96  P:   0

**PF(13): 0000101000

  COST: 0.34459985E 03   CPU:  32.450
  0000101000 0100101000 1011010011 001000001  S: 0.19160500E 05
  0000001000000000
  0000101000 0001101000 1110010011 110000001  S: 0.19160500E 05
  0000001000000000
  N:  96  P:   0
```

```
**PF(14): 0000101010

  COST: 0.41419995E 03   CPU:  34.404
  0000101010 0100101010 1011010001 001000000  S: 0.19520500E 05
  0000001000000000
  0000101010 0001101010 1110010001 110000000  S: 0.19520500E 05
  0000001000000000
  N:  64  R:   0

**PF(15): 0000101100

  COST: 0.42499976E 03   CPU:  36.357
  0000101100 0100101100 1011010011 001000001  S: 0.27560500E 05
  0000001000000000
  0000101100 0001101100 1110010011 110000001  S: 0.27560500E 05
  0000001000000000
  N:  64  R:   0

**PF(16): 0000110000

  COST: 0.36619971E 03   CPU:  39.190
  0000110000 0100110000 1011001011 001000001  S: 0.19160500E 05
  0000001000000000
  0000110000 0001110000 1110001011 110000001  S: 0.19160500E 05
  0000001000000000
  N:  96  R:   0

**PF(17): 0000110010

  COST: 0.43579980E 03   CPU:  41.140
  0000110010 0100110010 1011001001 001000000  S: 0.19520500E 05
  0000001000000000
  0000110010 0001110010 1110001001 110000000  S: 0.19520500E 05
  0000001000000000
  N:  64  R:   0

**PF(18): 0000110100

  COST: 0.45859985E 03   CPU:  43.082
  0000110100 0100110100 1011000011 001000001  S: 0.27560500E 05
  0000001000000000
  0000110100 0100110100 1011001011 001000001  S: 0.27560500E 05
  0000001000000000
  0000110100 0001110100 1110000011 110000001  S: 0.27560500E 05
  0000001000000000
  00[illegible] 0001110100 1110001011 110000001  S: 0.27560500E 05
  0000001000000000
  N:  64  R:   0
```

```
**PF(19): 0001000000

  COST: 0.31259985E 03   CPU:  48.855
  0001000000 0001000000 0110001111 010000111  S: 0.13360500E 05
  0000001000000000
  0001000000 0001000000 0110011011 010001001  S: 0.13360500E 05
  0000001000000000
  0001000000 0001000000 0110101111 010000111  S: 0.13360500E 05
  0000001000000000
  0001000000 0001000000 0110111011 010001001  S: 0.13360500E 05
  0000001000000000
  0001000000 0001000000 1100001111 100000111  S: 0.13360500E 05
  0000001000000000
  0001000000 0001000000 1100011011 100001001  S: 0.13360500E 05
  0000001000000000
  0001000000 0001000000 1100101111 100000111  S: 0.13360500E 05
  0000001000000000
  0001000000 0001000000 1100111011 100001001  S: 0.13360500E 05
  0000001000000000
  0001000000 0001000000 1110001111 110000111  S: 0.13360500E 05
  0000001000000000
  0001000000 0001000000 1110011011 110001001  S: 0.13360500E 05
  0000001000000000
  0001000000 0001000000 1110101111 110000111  S: 0.13360500E 05
  0000001000000000
  0001000000 0001000000 1110111011 110001001  S: 0.13360500E 05
  0000001000000000
  N: 192  R:   0
```

```
**PF(20): 0001000010

  COST: 0.38219971E 03   CPU:  53.397
  0001000010 0001000010 0110001101 010000100  S: 0.13720500E 05
  0000001000000000
  0001000010 0001000010 0110011001 010001000  S: 0.13720500E 05
  0000001000000000
  0001000010 0001000010 0110101101 010000100  S: 0.13720500E 05
  0000001000000000
  0001000010 0001000010 0110111001 010001000  S: 0.13720500E 05
  0000001000000000
  0001000010 0001000010 1100001101 100000100  S: 0.13720500E 05
  0000001000000000
  0001000010 0001000010 1100011001 100001000  S: 0.13720500E 05
  0000001000000000
  0001000010 0001000010 1100101101 100000100  S: 0.13720500E 05
  0000001000000000
  0001000010 0001000010 1100111001 100001000  S: 0.13720500E 05
  0000001000000000
  0001000010 0001000010 1110001101 110000100  S: 0.13720500E 05
  0000001000000000
  0001000010 0001000010 1110011001 110001000  S: 0.13720500E 05
  0000001000000000
  0001000010 0001000010 1110101101 110000100  S: 0.13720500E 05
  0000001000000000
  0001000010 0001000010 1110111001 110001000  S: 0.13720500E 05
  0000001000000000
  N: 144  R:   0
```

```
**PF(21): 0001000100

  COST: 0.40499976E 03   CPU:  57.851
  0001000100 0001000100 0110010011 010000001  S: 0.21760500E 05
  0000001000000000
  0001000100 0001000100 0110011011 010001001  S: 0.21760500E 05
  0000001000000000
  0001000100 0001000100 0110110011 010000001  S: 0.21760500E 05
  0000001000000000
  0001000100 0001000100 0110111011 010001001  S: 0.21760500E 05
  0000001000000000
  0001000100 0001000100 1100010011 100000001  S: 0.21760500E 05
  0000001000000000
  0001000100 0001000100 1100011011 100001001  S: 0.21760500E 05
  0000001000000000
  0001000100 0001000100 1100110011 100000001  S: 0.21760500E 05
  0000001000000000
  0001000100 0001000100 1100111011 100001001  S: 0.21760500E 05
  0000001000000000
  0001000100 0001000100 1110010011 110000001  S: 0.21760500E 05
  0000001000000000
  0001000100 0001000100 1110011011 110001001  S: 0.21760500E 05
  0000001000000000
  0001000100 0001000100 1110110011 110000001  S: 0.21760500E 05
  0000001000000000
  0001000100 0001000100 1110111011 110001001  S: 0.21760500E 05
  0000001000000000
  N: 144  R:   0

**PF(22): 0001001000

  COST: 0.34859985E 03   CPU:  62.324
  0001001000 0001001000 0110010011 010000001  S: 0.16960500E 05
  0000001000000000
  0001001000 0001001000 0110110011 010000001  S: 0.16960500E 05
  0000001000000000
  0001001000 0001001000 1100010011 100000001  S: 0.16960500E 05
  0000001000000000
  0001001000 0001001000 1100110011 100000001  S: 0.16960500E 05
  0000001000000000
  0001001000 0001001000 1110010011 110000001  S: 0.16960500E 05
  0000001000000000
  0001001000 0001001000 1110110011 110000001  S: 0.16960500E 05
  0000001000000000
  N: 144  R:   0
```

```
**PF(23): 0001001010

  COST: 0.41819995E 03   CPU:  65.301
  0001001010 0001001010 011001C001 010000000  S: 0.17320500E 05
  0000001000000000
  0001001010 0001001010 0110110001 010000000  S: 0.17320500E 05
  0000001000000000
  0001001010 0001001010 1100010001 100000000  S: 0.17320500E 05
  0000001000000000
  0001001010 0001001010 1100110001 100000000  S: 0.17320500E 05
  0000001000000000
  0001001010 0001001010 1110010001 110000000  S: 0.17320500E 05
  0000001000000000
  0001001010 0001001010 1110110001 110000000  S: 0.17320500E 05
  0000001000000000
  N:  96  R:   0

**PF(24): 0001001100

  COST: 0.42899976E 03   CPU:  68.261
  0001001100 0001001100 0110010011 010000001  S: 0.25360500E 05
  0000001000000000
  0001001100 0001001100 0110110011 010000001  S: 0.25360500E 05
  0000001000000000
  0001001100 0001001100 1100010011 100000001  S: 0.25360500E 05
  0000001000000000
  0001001100 0001001100 1100110011 100000001  S: 0.25360500E 05
  0000001000000000
  0001001100 0001001100 1110010011 110000001  S: 0.25360500F 05
  0000001000000000
  0001001100 0001001100 1110110011 110000001  S: 0.25360500E 05
  0000001000000000
  N:  96  R:   0

**PF(25): 0001010000

  COST: 0.37019971E 03   CPU:  72.560
  0001010000 0001010000 0110001011 010000001  S: 0.16960500E 05
  0000001000000000
  0001010000 0001010000 0110101011 010000001  S: 0.16960500E 05
  0000001000000000
  0001010000 0001010000 1100001011 100000001  S: 0.16960500E 05
  0000001000000000
  0001010000 0001010000 1100101011 100000001  S: 0.16960500E 05
  0000001000000000
  0001010000 0001010000 1110001011 110000001  S: 0.16960500E 05
  0000001000000000
  0001010000 0001010000 1110101011 110000001  S: 0.16960500E 05
  0000001000000000
  N: 144  R:   0
```

```
**PF(26): 0001010010

  COST: 0.43979980E 03   CPU:  75.517
  0001010010 0001010010 0110001001 010000000  S: 0.17320500E 05
  0000001000000000
  0001010010 0001010010 0110101001 010000000  S: 0.17320500E 05
  0000001000000000
  0001010010 0001010010 1100001001 100000000  S: 0.17320500E 05
  0000001000000000
  0001010010 0001010010 1100101001 100000000  S: 0.17320500E 05
  0000001000000000
  0001010010 0001010010 1110001001 110000000  S: 0.17320500E 05
  0000001000000000
  0001010010 0001010010 1110101001 110000000  S: 0.17320500E 05
  0000001000000000
  N:  96  R:   0

**PF(27): 0001010100

  COST: 0.46259985E 03   CPU:  78.462
  0001010100 0001010100 0110000011 010000001  S: 0.25360500E 05
  0000001000000000
  0001010100 0001010100 0110001011 010000001  S: 0.25360500E 05
  0000001000000000
  0001010100 0001010100 0110100011 010000001  S: 0.25360500E 05
  0000001000000000
  0001010100 0001010100 0110101011 010000001  S: 0.25360500E 05
  0000001000000000
  0001010100 0001010100 1100000011 100000001  S: 0.25360500E 05
  0000001000000000
  0001010100 0001010100 1100001011 100000001  S: 0.25360500E 05
  0000001000000000
  0001010100 0001010100 1100100011 100000001  S: 0.25360500E 05
  0000001000000000
  0001010100 0001010100 1100101011 100000001  S: 0.25360500E 05
  0000001000000000
  0001010100 0001010100 1110000011 110000001  S: 0.25360500E 05
  0000001000000000
  0001010100 0001010100 1110001011 110000001  S: 0.25360500E 05
  0000001000000000
  0001010100 0001010100 1110100011 110000001  S: 0.25360500E 05
  0000001000000000
  0001010100 0001010100 1110101011 110000001  S: 0.25360500E 05
  0000001000000000
  N:  96  R:   0

**PF(28): 0010000000

  COST: 0.30859985E 03   CPU:  81.600
  0010000000 0110000000 1001101111 000100111  S: 0.15560500E 05
  0000001000000000
  0010000000 0110000000 1001111011 000101001  S: 0.15560500E 05
  0000001000000000
  N:  96  R:   0
```

```
**PF(29): 0010000010

  COST: 0.37819971E 03   CPU:  84.049
  0010000010 0110000010 1001101101 000100100  S: 0.15920500E 05
  0000001000000000
  0010000010 0110000010 1001111001 000101000  S: 0.15920500E 05
  0000001000000000
  N:  80  R:   0

**PF(30): 0010000100

  COST: 0.38299976E 03   CPU:  86.394
  0010000100 0011000100 0100111011 000001001  S: 0.20960500E 05
  0000001000000000
  0010000100 0011000100 1100111011 100001001  S: 0.20960500E 05
  0000001000000000
  N:  80  R:   0

**PF(31): 0010001000

  COST: 0.32659985E 03   CPU:  89.261
  0010001000 0011001000 0100110011 000000001  S: 0.16160500E 05
  0000001000000000
  0010001000 0011001000 1100110011 100000001  S: 0.16160500E 05
  0000001000000000
  N:  96  R:   0

**PF(32): 0010001010

  COST: 0.39619995E 03   CPU:  91.211
  0010001010 0011001010 0100110001 000000000  S: 0.16520500E 05
  0000001000000000
  0010001010 0011001010 1100110001 100000000  S: 0.16520500E 05
  0000001000000000
  N:  64  R:   0

**PF(33): 0010001100

  COST: 0.40699976E 03   CPU:  93.119
  0010001100 0011001100 0100110011 000000001  S: 0.24560500E 05
  0000001000000000
  0010001100 0011001100 1100110011 100000001  S: 0.24560500E 05
  0000001000000000
  N:  64  R:   0

**PF(34): 0010010000

  COST: 0.34819971E 03   CPU:  95.969
  0010010000 0011010000 0100101011 000000001  S: 0.16160500E 05
  0000001000000000
  0010010000 0011010000 1100101011 100000001  S: 0.16160500E 05
  0000001000000000
  N:  96  R:   0
```

```
**PF(35): 0010010010

  COST: 0.41779980E 03   CPU:  97.915
  0010010010 0011010010 0100101001 000000000  S: 0.16520500E 05
  0000001000000000
  0010010010 0011010010 1100101001 100000000  S: 0.16520500E 05
  0000001000000000
  N:  64  R:   0

**PF(36): 0010010100

  COST: 0.44059985E 03   CPU:  99.839
  0010010100 0011010100 0100100011 000000001  S: 0.24560500E 05
  0000001000000000
  0010010100 0011010100 0100101011 000000001  S: 0.24560500E 05
  0000001000000000
  0010010100 0011010100 1100100011 100000001  S: 0.24560500E 05
  0000001000000000
  0010010100 0011010100 1100101011 100000001  S: 0.24560500E 05
  0000001000000000
  N:  64  R:   0

**PF(37): 0010100000

  COST: 0.31459985E 03   CPU: 102.694
  0010100000 0011100000 0100001111 000000111  S: 0.16160500E 05
  0000001000000000
  0010100000 0011100000 0100011011 000001001  S: 0.16160500E 05
  0000001000000000
  0010100000 0011100000 1100001111 100000111  S: 0.16160500E 05
  0000001000000000
  0010100000 0011100000 1100011011 100001001  S: 0.16160500E 05
  0000001000000000
  N:  96  R:   0

**PF(38): 0010100010

  COST: 0.38419971E 03   CPU: 104.879
  0010100010 0011100010 0100001101 000000100  S: 0.16520500E 05
  0000001000000000
  0010100010 0011100010 0100011001 000001000  S: 0.16520500E 05
  0000001000000000
  0010100010 0011100010 1100001101 100000100  S: 0.16520500E 05
  0000001000000000
  0010100010 0011100010 1100011001 100001000  S: 0.16520500E 05
  0000001000000000
  N:  72  R:   0
```

```
**PF(39): 0010100100

  COST: 0.40699976E 03   CPU: 107.052
  0010100100 0011100100 0100010011 000000001  S: 0.24560500E 05
  0000001000000000
  0010100100 0011100100 0100011011 000001001  S: 0.24560500E 05
  0000001000000000
  0010100100 0011100100 1100010011 100000001  S: 0.24560500E 05
  0000001000000000
  0010100100 0011100100 1100011011 100001001  S: 0.24560500E 05
  0000001000000000
  N:  72  R:   0

**PF(40): 0010101000

  COST: 0.35059985E 03   CPU: 109.207
  0010101000 0011101000 0100010011 000000001  S: 0.19760500E 05
  0000001000000000
  0010101000 0011101000 1100010011 100000001  S: 0.19760500E 05
  0000001000000000
  N:  72  R:   0

**PF(41): 0010101010

  COST: 0.42019995E 03   CPU: 110.650
  0010101010 0011101010 0100010001 000000000  S: 0.20120500E 05
  0000001000000000
  0010101010 0011101010 1100010001 100000000  S: 0.20120500E 05
  0000001000000000
  N:  48  R:   0

**PF(42): 0010101100

  COST: 0.43099976E 03   CPU: 112.115
  0010101100 0011101100 0100010011 000000001  S: 0.28160500E 05
  0000001000000000
  0010101100 0011101100 1100010011 100000001  S: 0.28160500E 05
  0000001000000000
  N:  48  R:   0

**PF(43): 0010110000

  COST: 0.37219971E 03   CPU: 114.256
  0010110000 0011110000 0100001011 000000001  S: 0.19760500E 05
  0000001000000000
  0010110000 0011110000 1100001011 100000001  S: 0.19760500E 05
  0000001000000000
  N:  72  R:   0
```

```
**PF(44): 0010110010

  COST: 0.44179980E 03   CPU: 115.698
  0010110010 0011110010 0100001001 000000000  S: 0.20120500E 05
  0000001000000000
  0010110010 0011110010 1100001001 100000000  S: 0.20120500E 05
  0000001000000000
  N:  48  R:   0

**PF(45): 0010110100

  COST: 0.46459985E 03   CPU: 117.153
  0010110100 0011110100 0100000011 000000001  S: 0.28160500E 05
  0000001000000000
  0010110100 0011110100 0100001011 000000001  S: 0.28160500E 05
  0000001000000000
  0010110100 0011110100 1100000011 100000001  S: 0.28160500E 05
  0000001000000000
  0010110100 0011110100 1100001011 100000001  S: 0.28160500E 05
  0000001000000000
  N:  48  R:   0

**PF(46): 0011000000

  COST: 0.31659985E 03   CPU: 121.024
  0011000000 0011000000 0100001111 000000111  S: 0.13960500E 05
  0000001000000000
  0011000000 0011000000 0100011011 000001001  S: 0.13960500E 05
  0000001000000000
  0011000000 0011000000 0100101111 000000111  S: 0.13960500E 05
  0000001000000000
  0011000000 0011000000 0100111011 000001001  S: 0.13960500E 05
  0000001000000000
  0011000000 0011000000 1100001111 100000111  S: 0.13960500E 05
  0000001000000000
  0011000000 0011000000 1100011011 100001001  S: 0.13960500E 05
  0000001000000000
  0011000000 0011000000 1100101111 100000111  S: 0.13960500E 05
  0000001000000000
  0011000000 0011000000 1100111011 100001001  S: 0.13960500E 05
  0000001000000000
  N: 128  R:   0
```

```
**PF(47): 0011000010

   COST: 0.38619971E 03   CPU: 124.056
   0011000010 0011000010 0100001101 000000100  S: 0.14320500E 05
   0000001000000000
   0011000010 0011000010 0100011001 000001000  S: 0.14320500E 05
   0000001000000000
   0011000010 0011000010 0100101101 000000100  S: 0.14320500E 05
   0000001000000000
   0011000010 0011000010 0100111001 000001000  S: 0.14320500E 05
   0000001000000000
   0011000010 0011000010 1100001101 100000100  S: 0.14320500E 05
   0000001000000000
   0011000010 0011000010 1100011001 100001000  S: 0.14320500E 05
   0000001000000000
   0011000010 0011000010 1100101101 100000100  S: 0.14320500E 05
   0000001000000000
   0011000010 0011000010 1100111001 100001000  S: 0.14320500E 05
   0000001000000000
   N:  96  R:   0

**PF(48): 0011000100

   COST: 0.40899976E 03   CPU: 127.033
   0011000100 0011000100 0100010011 000000001  S: 0.22360500E 05
   0000001000000000
   0011000100 0011000100 0100011011 000001001  S: 0.22360500E 05
   0000001000000000
   0011000100 0011000100 0100110011 000000001  S: 0.22360500E 05
   0000001000000000
   0011000100 0011000100 0100111011 000001001  S: 0.22360500E 05
   0000001000000000
   0011000100 0011000100 1100010011 100000001  S: 0.22360500E 05
   0000001000000000
   0011000100 0011000100 1100011011 100001001  S: 0.22360500E 05
   0000001000000000
   0011000100 0011000100 1100110011 100000001  S: 0.22360500E 05
   0000001000000000
   0011000100 0011000100 1100111011 100001001  S: 0.22360500E 05
   0000001000000000
   N:  96  R:   0

**PF(49): 0011001000

   COST: 0.35259985E 03   CPU: 129.972
   0011001000 0011001000 0100010011 000000001  S: 0.17560500E 05
   0000001000000000
   0011001000 0011001000 0100110011 000000001  S: 0.17560500E 05
   0000001000000000
   0011001000 0011001000 1100010011 100000001  S: 0.17560500E 05
   0000001000000000
   0011001000 0011001000 1100110011 100000001  S: 0.17560500E 05
   0000001000000000
   N:  96  P:   0
```

```
**PF(50): 0011001010

  COST: 0.42219995E 03   CPU: 131.965
  0011001010 0011001010 0100010001 000000000  S: 0.17920500E 05
  0000001000000000
  0011001010 0011001010 0100110001 000000000  S: 0.17920500E 05
  0000001000000000
  0011001010 0011001010 1100010001 100000000  S: 0.17920500E 05
  0000001000000000
  0011001010 0011001010 1100110001 100000000  S: 0.17920500E 05
  0000001000000000
  N:  64  R:   0

**PF(51): 0011001100

  COST: 0.43299976E 03   CPU: 133.921
  0011001100 0011001100 0100010011 000000001  S: 0.25960500E 05
  0000001000000000
  0011001100 0011001100 0100110011 000000001  S: 0.25960500E 05
  0000001000000000
  0011001100 0011001100 1100010011 100000001  S: 0.25960500E 05
  0000001000000000
  0011001100 0011001100 1100110011 100000001  S: 0.25960500E 05
  0000001000000000
  N:  64  R:   0

**PF(52): 0011010000

  COST: 0.37419971E 03   CPU: 136.840
  0011010000 0011010000 0100001011 000000001  S: 0.17560500E 05
  0000001000000000
  0011010000 0011010000 0100101011 000000001  S: 0.17560500E 05
  0000001000000000
  0011010000 0011010000 1100001011 100000001  S: 0.17560500E 05
  0000001000000000
  0011010000 0011010000 1100101011 100000001  S: 0.17560500E 05
  0000001000000000
  N:  96  R:   0

**PF(53): 0011010010

  COST: 0.44379980E 03   CPU: 138.822
  0011010010 0011010010 0100001001 000000000  S: 0.17920500E 05
  0000001000000000
  0011010010 0011010010 0100101001 000000000  S: 0.17920500E 05
  0000001000000000
  0011010010 0011010010 1100001001 100000000  S: 0.17920500E 05
  0000001000000000
  0011010010 0011010010 1100101001 100000000  S: 0.17920500E 05
  0000001000000000
  N:  64  R:   0
```

```
**PF(54): 0011010100

   COST: 0.46659985E 03   CPU: 140.818
   0011010100 0011010100 0100000011 000000001  S: 0.25960500E 05
   0000001000000000
   0011010100 0011010100 0100001011 000000001  S: 0.25960500E 05
   0000001000000000
   0011010100 0011010100 0100100011 000000001  S: 0.25960500E 05
   0000001000000000
   0011010100 0011010100 0100101011 000000001  S: 0.25960500E 05
   0000001000000000
   0011010100 0011010100 1100000011 100000001  S: 0.25960500E 05
   0000001000000000
   0011010100 0011010100 1100001011 100000001  S: 0.25960500E 05
   0000001000000000
   0011010100 0011010100 1100100011 100000001  S: 0.25960500E 05
   0000001000000000
   0011010100 0011010100 1100101011 100000001  S: 0.25960500E 05
   0000001000000000
   N:  64  R:   0

**PF(55): 0100000000

   COST: 0.31259985E 03   CPU: 145.736
   0100000000 0100000000 0001101111 000100111  S: 0.13360500E 05
   0000001000000000
   0100000000 0100000000 0001111011 000101001  S: 0.13360500E 05
   0000001000000000
   0100000000 0100000000 0011001111 001000111  S: 0.13360500E 05
   0000001000000000
   0100000000 0100000000 0011011011 001001001  S: 0.13360500E 05
   0000001000000000
   0100000000 0100000000 0011101111 001100111  S: 0.13360500E 05
   0000001000000000
   0100000000 0100000000 0011111011 001101001  S: 0.13360500E 05
   0000001000000000
   0100000000 0100000000 1001101111 000100111  S: 0.13360500E 05
   0000001000000000
   0100000000 0100000000 1001111011 000101001  S: 0.13360500E 05
   0000001000000000
   0100000000 0100000000 1011001111 001000111  S: 0.13360500E 05
   0000001000000000
   0100000000 0100000000 1011011011 001001001  S: 0.13360500E 05
   0000001000000000
   0100000000 0100000000 1011101111 001100111  S: 0.13360500E 05
   0000001000000000
   0100000000 0100000000 1011111011 001101001  S: 0.13360500E 05
   0000001000000000
   N: 160  R:   0
```

```
**PF(56): 0100000010

  COST: 0.38219971E 03   CPU: 149.804
  0100000010 0100000010 0001101101 000100100  S: 0.13720500E 05
  0000001000000000
  0100000010 0100000010 0001111001 000101000  S: 0.13720500E 05
  0000001000000000
  0100000010 0100000010 0011001101 001000100  S: 0.13720500E 05
  0000001000000000
  0100000010 0100000010 0011011001 001001000  S: 0.13720500E 05
  0000001000000000
  0100000010 0100000010 0011101101 001100100  S: 0.13720500E 05
  0000001000000000
  0100000010 0100000010 0011111001 001101000  S: 0.13720500E 05
  0000001000000000
  0100000010 0100000010 1001101101 000100100  S: 0.13720500E 05
  0000001000000000
  0100000010 0100000010 1001111001 000101000  S: 0.13720500E 05
  0000001000000000
  0100000010 0100000010 1011001101 001000100  S: 0.13720500E 05
  0000001000000000
  0100000010 0100000010 1011011001 001001000  S: 0.13720500E 05
  0000001000000000
  0100000010 0100000010 1011101101 001100100  S: 0.13720500E 05
  0000001000000000
  0100000010 0100000010 1011111001 001101000  S: 0.13720500E 05
  0000001000000000
  N: 128  R:    0

**PF(57): 0100000100

  COST: 0.38699976E 03   CPU: 153.735
  0100000100 0101000100 0010111011 000001001  S: 0.20960500E 05
  0000001000000000
  0100000100 0101000100 1010111011 000001001  S: 0.20960500E 05
  0000001000000000
  N: 128  R:    0

**PF(58): 0100001000

  COST: 0.33059985E 03   CPU: 157.969
  0100001000 0101001000 0010110011 000000001  S: 0.16160500E 05
  0000001000000000
  0100001000 0101001000 1010110011 000000001  S: 0.16160500E 05
  0000001000000000
  N: 144  R:    0
```

```
**PF(59): 0100001010

  COST: 0.40019995E 03   CPU: 160.807
  0100001010 0101001010 0010110001 000000000  S: 0.16520500E 05
  0000001000000000
  0100001010 0101001010 1010110001 000000000  S: 0.16520500E 05
  0000001000000000
  N:  96  P:    0

**PF(60): 0100001100

  COST: 0.41099976E 03   CPU: 163.644
  0100001100 0101001100 0010110011 000000001  S: 0.24560500E 05
  0000001000000000
  0100001100 0101001100 1010110011 000000001  S: 0.24560500E 05
  0000001000000000
  N:  96  P:    0

**PF(61): 0100010000

  COST: 0.35219971E 03   CPU: 167.862
  0100010000 0101010000 0010101011 000000001  S: 0.16160500E 05
  0000001000000000
  0100010000 0101010000 1010101011 000000001  S: 0.16160500E 05
  0000001000000000
  N: 144  P:    0

**PF(62): 0100010010

  COST: 0.42179980E 03   CPU: 170.689
  0100010010 0101010010 0010101001 000000000  S: 0.16520500E 05
  0000001000000000
  0100010010 0101010010 1010101001 000000000  S: 0.16520500E 05
  0000001000000000
  N:  96  P:    0

**PF(63): 0100010100

  COST: 0.44459985E 03   CPU: 173.470
  0100010100 0101010100 0010100011 000000001  S: 0.24560500E 05
  0000001000000000
  0100010100 0101010100 0010101011 000000001  S: 0.24560500E 05
  0000001000000000
  0100010100 0101010100 1010100011 000000001  S: 0.24560500E 05
  0000001000000000
  0100010100 0101010100 1010101011 000000001  S: 0.24560500E 05
  0000001000000000
  N:  96  P:    0
```

```
**PF(64): 0100100000

   COST: 0.31859985E 03   CPU: 177.251
   0100100000 0101100000 0010001111 000000111  S: 0.16160500E 05
   0000001000000000
   0100100000 0101100000 0010011011 000001001  S: 0.16160500E 05
   0000001000000000
   0100100000 0101100000 1010001111 000000111  S: 0.1616050[illegible]E 05
   0000001000000000
   0100100000 0101100000 1010011011 000001001  S: 0.16160500E 05
   0000001000000000
   N: 128  P:   0

**PF(65): 0100100010

   COST: 0.38819971E 03   CPU: 180.124
   0100100010 0101100010 0010001101 000000100  S: 0.16520500E 05
   0000001000000000
   0100100010 0101100010 0010011001 000001000  S: 0.16520500E 05
   0000001000000000
   0100100010 0101100010 1010001101 000000100  S: 0.16520500E 05
   0000001000000000
   0100100010 0101100010 1010011001 000001000  S: 0.16520500E 05
   0000001000000000
   N:  96  P:   0

**PF(66): 0100100100

   COST: 0.41099976E 03   CPU: 182.929
   0100100100 0101100100 0010010011 000000001  S: 0.24560500E 05
   0000001000000000
   0100100100 0101100100 0010011011 000001001  S: 0.24560500E 05
   0000001000000000
   0100100100 0101100100 1010010011 000000001  S: 0.24560500E 05
   0000001000000000
   0100100100 0101100100 1010011011 000001001  S: 0.24560500E 05
   0000001000000000
   N:  96  P:   0

**PF(67): 0100101000

   COST: 0.35459985E 03   CPU: 185.793
   0100101000 0101101000 0010010011 000000001  S: 0.19760500E 05
   0000001000000000
   0100101000 0101101000 1010010011 000000001  S: 0.19760500E 05
   0000001000000000
   N:  96  P:   0
```

```
**PF(68): 0100101010

  COST: 0.42419995E 03   CPU: 187.687
  0100101010 0101101010 0010010001 000000000  S: 0.20120500E 05
  0000001000000000
  0100101010 0101101010 1010010001 000000000  S: 0.20120500E 05
  0000001000000000
  N:  64  R:   0

**PF(69): 0100101100

  COST: 0.43499976E 03   CPU: 189.593
  0100101100 0101101100 0010010011 000000001  S: 0.28160500E 05
  0000001000000000
  0100101100 0101101100 1010010011 000000001  S: 0.28160500E 05
  0000001000000000
  N:  64  R:   0

**PF(70): 0100110000

  COST: 0.37619971E 03   CPU: 192.402
  0100110000 0101110000 0010001011 000000001  S: 0.19760500E 05
  0000001000000000
  0100110000 0101110000 1010001011 000000001  S: 0.19760500E 05
  0000001000000000
  N:  96  R:   0

**PF(71): 0100110010

  COST: 0.44579980E 03   CPU: 194.287
  0100110010 0101110010 0010001001 000000000  S: 0.20120500E 05
  0000001000000000
  0100110010 0101110010 1010001001 000000000  S: 0.20120500E 05
  0000001000000000
  N:  64  R:   0

**PF(72): 0100110100

  COST: 0.46859985E 03   CPU: 196.172
  0100110100 0101110100 0010000011 000000001  S: 0.28160500E 05
  0000001000000000
  0100110100 0101110100 0010001011 000000001  S: 0.28160500E 05
  0000001000000000
  0100110100 0101110100 1010000011 000000001  S: 0.28160500E 05
  0000001000000000
  0100110100 0101110100 1010001011 000000001  S: 0.28160500E 05
  0000001000000000
  N:  64  R:   0
```

```
**PF(73): 0101000000

  COST: 0.32259985E 03   CPU: 199.978
  0101000000 0101000000 0000001111 000000111  S: 0.13960500E 05
  0000001000000000
  0101000000 0101000000 0000011011 000001001  S: 0.13960500E 05
  0000001000000000
  0101000000 0101000000 0000101111 000000111  S: 0.13960500E 05
  0000001000000000
  0101000000 0101000000 0000111011 000001001  S: 0.13960500E 05
  0000001000000000
  0101000000 0101000000 0010001111 000000111  S: 0.13960500E 05
  0000001000000000
  0101000000 0101000000 0010011011 000001001  S: 0.13960500E 05
  0000001000000000
  0101000000 0101000000 0010101111 000000111  S: 0.13960500E 05
  0000001000000000
  0101000000 0101000000 0010111011 000001001  S: 0.13960500E 05
  0000001000000000
  0101000000 0101000000 1000001111 000000111  S: 0.13960500E 05
  0000001000000000
  0101000000 0101000000 1000011011 000001001  S: 0.13960500E 05
  0000001000000000
  0101000000 0101000000 1000101111 000000111  S: 0.13960500E 05
  0000001000000000
  0101000000 0101000000 1000111011 000001001  S: 0.13960500E 05
  0000001000000000
  0101000000 0101000000 1010001111 000000111  S: 0.13960500E 05
  0000001000000000
  0101000000 0101000000 1010011011 000001001  S: 0.13960500E 05
  0000001000000000
  0101000000 0101000000 1010101111 000000111  S: 0.13960500E 05
  0000001000000000
  0101000000 0101000000 1010111011 000001001  S: 0.13960500E 05
  0000001000000000
  N: 128  P:    0
```

```
**PF(74): 0101000010

 COST: 0.39219971E 03   CPU: 203.140
 0101000010 0101000010 0000001101 000000100  S: 0.14320500E 05
 0000001000000000
 0101000010 0101000010 0000011001 000001000  S: 0.14320500E 05
 0000001000000000
 0101000010 0101000010 0000101101 000000100  S: 0.14320500E 05
 0000001000000000
 0101000010 0101000010 0000111001 000001000  S: 0.14320500E 05
 0000001000000000
 0101000010 0101000010 0010001101 000000100  S: 0.14320500E 05
 0000001000000000
 0101000010 0101000010 0010011001 000001000  S: 0.14320500E 05
 0000001000000000
 0101000010 0101000010 0010101101 000000100  S: 0.14320500E 05
 0000001000000000
 0101000010 0101000010 0010111001 000001000  S: 0.14320500E 05
 0000001000000000
 0101000010 0101000010 1000001101 000000100  S: 0.14320500E 05
 0000001000000000
 0101000010 0101000010 1000011001 000001000  S: 0.14320500E 05
 0000001000000000
 0101000010 0101000010 1000101101 000000100  S: 0.14320500E 05
 0000001000000000
 0101000010 0101000010 1000111001 000001000  S: 0.14320500E 05
 0000001000000000
 0101000010 0101000010 1010001101 000000100  S: 0.14320500E 05
 0000001000000000
 0101000010 0101000010 1010011001 000001000  S: 0.14320500E 05
 0000001000000000
 0101000010 0101000010 1010101101 000000100  S: 0.14320500E 05
 0000001000000000
 0101000010 0101000010 1010111001 000001000  S: 0.14320500E 05
 0000001000000000
 N:  96  R:   0
```

```
**PF(75): 0101000100

 COST: 0.41499976E 03   CPU: 206.257
 0101000100 0101000100 0000010011 000000001  S: 0.22360500E 05
 0000001000000000
 0101000100 0101000100 0000011011 000001001  S: 0.22360500E 05
 0000001000000000
 0101000100 0101000100 0000110011 000000001  S: 0.22360500E 05
 0000001000000000
 0101000100 0101000100 0000111011 000001001  S: 0.22360500E 05
 0000001000000000
 0101000100 0101000100 0010010011 000000001  S: 0.22360500E 05
 0000001000000000
 0101000100 0101000100 0010011011 000001001  S: 0.22360500E 05
 0000001000000000
 0101000100 0101000100 0010110011 000000001  S: 0.22360500E 05
 0000001000000000
 0101000100 0101000100 0010111011 000001001  S: 0.22360500E 05
 0000001000000000
 0101000100 0101000100 1000010011 000000001  S: 0.22360500E 05
 0000001000000000
 0101000100 0101000100 1000011011 000001001  S: 0.22360500E 05
 0000001000000000
 0101000100 0101000100 1000110011 000000001  S: 0.22360500E 05
 0000001000000000
 0101000100 0101000100 1000111011 000001001  S: 0.22360500E 05
 0000001000000000
 0101000100 0101000100 1010010011 000000001  S: 0.22360500E 05
 0000001000000000
 0101000100 0101000100 1010011011 000001001  S: 0.22360500E 05
 0000001000000000
 0101000100 0101000100 1010110011 000000001  S: 0.22360500E 05
 0000001000000000
 0101000100 0101000100 1010111011 000001001  S: 0.22360500E 05
 0000001000000000
 N:  96  R:   0
```

```
**PF(76): 0101001000

  COST: 0.35859985E 03   CPU: 209.414
  0101001000 0101001000 0000010011 000000001  S: 0.17560500E 05
  0000001000000000
  0101001000 0101001000 0000110011 000000001  S: 0.17560500E 05
  0000001000000000
  0101001000 0101001000 0010010011 000000001  S: 0.17560500E 05
  0000001000000000
  0101001000 0101001000 0010110011 000000001  S: 0.17560500E 05
  0000001000000000
  0101001000 0101001000 1000010011 000000001  S: 0.17560500E 05
  0000001000000000
  0101001000 0101001000 1000110011 000000001  S: 0.17560500E 05
  0000001000000000
  0101001000 0101001000 1010010011 000000001  S: 0.17560500E 05
  0000001000000000
  0101001000 0101001000 1010110011 000000001  S: 0.17560500E 05
  0000001000000000
  N:  96  R:   0

**PF(77): 0101001010

  COST: 0.42819995E 03   CPU: 211.485
  0101001010 0101001010 0000010001 000000000  S: 0.17920500E 05
  0000001000000000
  0101001010 0101001010 0000110001 000000000  S: 0.17920500E 05
  0000001000000000
  0101001010 0101001010 0010010001 000000000  S: 0.17920500E 05
  0000001000000000
  0101001010 0101001010 0010110001 000000000  S: 0.17920500E 05
  0000001000000000
  0101001010 0101001010 1000010001 000000000  S: 0.17920500E 05
  0000001000000000
  0101001010 0101001010 1000110001 000000000  S: 0.17920500E 05
  0000001000000000
  0101001010 0101001010 1010010001 000000000  S: 0.17920500E 05
  0000001000000000
  0101001010 0101001010 1010110001 000000000  S: 0.17920500E 05
  0000001000000000
  N:  64  R:   0
```

```
**PF(78): 0101001100

  COST: 0.43899976E 03   CPU: 213.524
  0101001100 0101001100 0000010011 000000001  S: 0.25960500E 05
  0000001000000000
  0101001100 0101001100 0000110011 000000001  S: 0.25960500E 05
  0000001000000000
  0101001100 0101001100 0010010011 000000001  S: 0.25960500E 05
  0000001000000000
  0101001100 0101001100 0010110011 000000001  S: 0.25960500E 05
  0000001000000000
  0101001100 0101001100 1000010011 000000001  S: 0.25960500E 05
  0000001000000000
  0101001100 0101001100 1000110011 000000001  S: 0.25960500E 05
  0000001000000000
  0101001100 0101001100 1010010011 000000001  S: 0.25960500E 05
  0000001000000000
  0101001100 0101001100 1010110011 000000001  S: 0.25960500E 05
  0000001000000000
  N:  64  R:   0

**PF(79): 0101010000

  COST: 0.38019971E 03   CPU: 216.473
  0101010000 0101010000 0000001011 000000001  S: 0.17560500E 05
  0000001000000000
  0101010000 0101010000 0000101011 000000001  S: 0.17560500E 05
  0000001000000000
  0101010000 0101010000 0010001011 000000001  S: 0.17560500E 05
  0000001000000000
  0101010000 0101010000 0010101011 000000001  S: 0.17560500E 05
  0000001000000000
  0101010000 0101010000 1000001011 000000001  S: 0.17560500E 05
  0000001000000000
  0101010000 0101010000 1000101011 000000001  S: 0.17560500E 05
  0000001000000000
  0101010000 0101010000 1010001011 000000001  S: 0.17560500E 05
  0000001000000000
  0101010000 0101010000 1010101011 000000001  S: 0.17560500E 05
  0000001000000000
  N:  96  R:   0
```

```
**PF(80): 0101010010

  COST: 0.44979980E 03   CPU: 218.531
  0101010010 0101010010 0000001001 000000000  S: 0.17920500E 05
  0000001000000000
  0101010010 0101010010 0000101001 000000000  S: 0.17920500E 05
  0000001000000000
  0101010010 0101010010 0010001001 000000000  S: 0.17920500E 05
  0000001000000000
  0101010010 0101010010 0010101001 000000000  S: 0.17920500E 05
  0000001000000000
  0101010010 0101010010 1000001001 000000000  S: 0.17920500E 05
  0000001000000000
  0101010010 0101010010 1000101001 000000000  S: 0.17920500E 05
  0000001000000000
  0101010010 0101010010 1010001001 000000000  S: 0.17920500E 05
  0000001000000000
  0101010010 0101010010 1010101001 000000000  S: 0.17920500E 05
  0000001000000000
  N:   64  R:    0
```

```
**PF(81): 0101010100

  COST: 0.47259985E 03   CPU: 220.589
  0101010100 0101010100 0000000011 000000001  S: 0.25960500E 05
  0000001000000000
  0101010100 0101010100 0000001011 000000001  S: 0.25960500E 05
  0000001000000000
  0101010100 0101010100 0000100011 000000001  S: 0.25960500E 05
  0000001000000000
  0101010100 0101010100 0000101011 000000001  S: 0.25960500E 05
  0000001000000000
  0101010100 0101010100 0010000011 000000001  S: 0.25960500E 05
  0000001000000000
  0101010100 0101010100 0010001011 000000001  S: 0.25960500E 05
  0000001000000000
  0101010100 0101010100 0010100011 000000001  S: 0.25960500E 05
  0000001000000000
  0101010100 0101010100 0010101011 000000001  S: 0.25960500E 05
  0000001000000000
  0101010100 0101010100 1000000011 000000001  S: 0.25960500E 05
  0000001000000000
  0101010100 0101010100 1000001011 000000001  S: 0.25960500E 05
  0000001000000000
  0101010100 0101010100 1000100011 000000001  S: 0.25960500E 05
  0000001000000000
  0101010100 0101010100 1000101011 000000001  S: 0.25960500E 05
  0000001000000000
  0101010100 0101010100 1010000011 000000001  S: 0.25960500E 05
  0000001000000000
  0101010100 0101010100 1010001011 000000001  S: 0.25960500E 05
  0000001000000000
  0101010100 0101010100 1010100011 000000001  S: 0.25960500E 05
  0000001000000000
  0101010100 0101010100 1010101011 000000001  S: 0.25960500E 05
  0000001000000000
  N:  64  R:   0

PF(28) CONTAINS BEST SOLUTION(S).   7232 SOLUTIONS CONSIDERED.

   MEAN COST OVER ALL SOLUTIONS:   0.46321021E 03
   MAXIMUM COST ENCOUNTERED:       0.67499951E 03

********** SOLUTION TERMINATED **********
```

## BIBLIOGRAPHY

1. Ash, W. L. and E. H. Sibley, "TRAMP: An Interpretive Associative Processor with Deductive Capabilities", Proc. ACM Nat. Conf. (1968), pp. 143-156.

2. Balas, Egon, "An Additive Algorithm for Solving Linear Programs with Zero-One Variables", Operations Research, 13: 4, July-August 1965, pp. 517-546.

3. Balinski, M. L., "Integer Programming: Methods, Uses, Computation", Management Science, 12: 3, November 1965, pp. 253-313.

4. Berkeley, Edmund C., "The Programming Language LISP: An Introduction and Appraisal", Computers and Automation, September 1964, pp. 16-23.

5. Bobrow, Daniel G. and Bertram Raphael. "A Comparison of List-Processing Computer Languages", CACM, 7: 4, April 1964, pp. 231-240.

6. Brewer, S., "Data Base or Data Maze", Proc. ACM Nat. Conf. (1968), pp. 623-630.

7. Busacker, Robert G. and Thomas L. Saaty, Finite Graphs and Networks: An Introduction with Applications (New York: McGraw-Hill, 1965).

8. Carr, J. W. III, "Recursive Subscripting Compilers and List-Type Memories", CACM, 2: 2, February 1959, pp. 4-6.

9. Chapin, N., "A Deeper Look at Data", Proc. ACM Nat. Conf. (1968), pp. 631-638.

10. Chapin, N., "Common File Organization Techniques Compared", Proc. FJCC (1969), pp. 413-422.

11. Childs, David L., "Description of a Set-Theoretic Data Structure", Proc. FJCC (1968), pp. 557-564.

12. Codd, E. F., "A Relational Model of Data for Large Shared Data Banks", CACM, 13: 6, June 1970, pp. 377-387.

13. Coffman, E. G. and J. Eve, "File Structures Using Hashing Functions", CACM, 13: 7, July 1970, pp. 427-432.

14. Comfort, W. T., "Multiword List Items", CACM, 7: 6, June 1964, pp. 357-362.

15. Conway, R. W. et al, "CLP - The Cornell List Processor", CACM, 8: 4, April 1965, pp. 215-216.

16. Cooper, D. C. and H. Whitfield, ALP: An Autocode List-Processing Language", Computer Journal, April 1962, pp. 28-32.

17. D'Imperio, Mary, "Data Structures and their Representations in Storage: Parts I and II", Reprinted from NSA Technical Journal, 9: 3 and 4, 1964, pp. 59-81, pp. 7-54.

18. D'Imperio, Mary, "Data Structures and their Representation in Storage", Annual Review in Automatic Programming, 5, 1969, pp. 1-75

19. Dodd, G. G., "Elements of Data Management Systems", Computing Surveys, 1: 2, June 1969, pp. 117-133.

20. Feldman, J. A., "Aspects of Associative Processing", Technical Note 1965-13, MIT Lincoln Laboratory, Lexington, Massachusetts, 1965.

21. Feldman, Jerome A. and Paul D. Rovner, "An Algol-Based Associative Language", CACM, 12: 8, August 1969, pp. 439-449.

22. Gass, Saul I, Linear Programming: Methods and Applications, (New York: McGraw-Hill, 1964).

23. Gelernter, H. et al, "A Fortran-Compiled List-Processing Language", JACM, 7. 2, April 1960, pp. 87-101.

24. Golomb, S. W. and L. D. Baumert, "Backtrack Programming", JACM, 12: 4, October 1965, pp. 516-524.

25. Gorry, G. A., "A System for Computer-Aided Diagnosis", Doctoral Thesis, MAC-TR-44, Massachusetts Institute of Technology, September 1967.

26. Gorry, G. A. and G. O. Barnett, "Sequential Diagnosis by Computer", JAMA, 205, September 1968, pp 849-854.

27. Gustafson, D. H. et al, "Subjective Probabilities in Medical Diagnosis", IEEE Trans. Man-Machine Systems, MMS-10: 3, September 1969, pp. 61-65.

28. Halmos, Paul R., Naive Set Theory, (New York: Van Nostrand, 1960).

29. Hammer (Ivanescu), P. L. and S. Rudeanu, Boolean Methods in Operations Research and Related Areas, (New York: Springer-Verlag, 1968).

30. Hansen, Wilfred J., "Compact List Representation: Definition, Garbage Collection, and System Implementation", CACM, 12: 9, September 1969, pp. 499-507.

31. Harrison, M., "BALM - An Extendable List-Processing Language", Proc. SJCC (1970), pp. 507-511.

32. Hellerman, H., "Addressing Multidimensional Arrays", CACM, 5: 4, April 1962, pp. 205-207.

33. Hsiao, David and Frank Harary, "A Formal System for Information Retrieval from Files", CACM, 13: 2, February 1970, pp. 67-73.

34. IBM Corporation, "IBM System/360 Model 67 Functional Characteristics", Form A27-2719-0, 1967.

35. Ivanescu, P. L., "Pseudo-Boolean Programming and Applications", Lecture Notes in Mathematics, 9 (Berlin: Springer-Verlag, 1965).

36. Ivanescu, P. L. and S. Rudeanu, "Pseudo Boolean Methods for Bivalent Programming", Lecture Notes in Mathematics, 23 (Berlin: Springer-Verlag, 1966).

37. Iverson, K. E., "A Programming Notation for Trees", IBM Corp. Research Report, RC-390, January 1961.

38. Iverson, K. E., A Programming Language, (New York: Wiley, 1962).

39. Jacquez, J. A. (Ed), The Diagnostic Process, Proc. Conf. held at The University of Michigan, May 1963 (Ann Arbor: Malloy Lithographing, 1964).

40. Kleinmutz, B., Clinical Information Processing by Computer (New York: Holt, Rinehart and Winston, 1969).

41. Knowlton, Kenneth C., A Programmer's Description of $L^6$, Bell Telephone Laboratories' Low-Level Linked List Language, February 1966, Murray Hill, N. J.

42. Knowlton, Kenneth C., "A Programmer's Description of $L^6$", CACM, 9: 8, August 1966, pp. 616-62[illegible].

43. Knuth, Donald E., The Art of Computer Programming, Volume 1, Fundamental Algorithms (Reading, Massachusetts: Addison-Wesley, 1968).

44. Lang, C. A. and J. C. Gray, "ASP - A Ring Implemented Associative Structure Package", CACM, 11: 8, August 1968, pp. 550-555.

45. Lawler, E. L. and M. D. Bell, "A Method for Solving Discrete Optimization Problems", Operations Research, 14: 6, November-December 1966, pp. 1098-1112.

46. Lawler, E. L. and D. E. Wood, "Branch-and-Bound Methods: A Survey", Operations Research, 14: 4, July-August 1966, pp. 699-719.

47. Lawson, Harold W., Jr., "PL/I List Processing", CACM, 10: 6, June 1967, pp. 358-367.

48. Ledley, R. S., Use of Computers in Biology and Medicine (New York: McGraw-Hill, 1965).

49. Lipovski, G., "The Architecture of a Large Associative Processor", Proc. SJCC (1970), pp. 38[illegible]-396.

50. Lipschutz, Seymour, Theory and Problems of Set Theory and Related Topics, Schaum's Outline Series, (New York: Schaum, 1964).

51. Lowe, Thomas J., "The Influence of Data-Base Characteristics and Usage on Direct-Access File Organization", JACM, 15: 4, October 1968, pp. 535-548.

52. Lusted, L. B., Introduction to Medical Decision Making (Springfield: Thomas, 1969).

53. Madnick, Stuart E., "Strong Processing Techniques", CACM, 10: 7, July 1967, pp. 420-424.

54. McCarthy, John, "Recursive Functions of Symbolic Expressions and their Computation by Machine, Part I", CACM, 3: 4, April 1960, pp. 184-195.

55. McCuskey, William A., "On Automatic Design of Data Organization", Proc. FJCC (1970), pp. 187-199.

56. McGee, William C., "File Structures for Generalized Data Management", Proc. IFIP Congress (1968), Applications 1, Booket F, pp. 68-73.

57. Mealy, George H., "Another Look at Data", Proc. FJCC (1967), pp. 525-534.

58. Morris, Robert, "Scatter Storage Techniques", CACM, 11: 1, January 1968, pp. 38-44.

59. Newell, A. and J. C. Shaw, "Programming the Logic Theory Machine", Proc. Western Joint Computer Conference, 11, February 1957, pp. 230-240.

60. Newell, A. and F. M. Tonge, "An Introduction to Information Processing Language V", CACM, 3: 4, April 1960, pp. 205-211.

61. Newman, William M., A System for Interactive Graphical Programming, Computer Tech. Group Report 67/7, Centre for Computing and Automation, Imperial College, October 1967.

62. Newman, William M., The ASP-7 Ring-Structure Processor, Computer Tech. Group Report 67/8, Centre for Computing and Automation, Imperial College, October 1967.

63. Patt, Yale N., "Variable Length Tree Structures Having Minimum Average Search Time", CACM, 12: 2, February 1969, pp. 72-76.

64. Perlis, A. J. and Charles Thornton, "Symbol Manipulation by Threaded Lists", CACM, 3: 4, April 1960, pp. 195-204.

65. Ross, Douglas T., "A Generalized Technique for Symbol Manipulation and Numerical Calculation", CACM, 4: 3, March 1961, pp. 147-150.

66. Roth, Richard H., "An Approach to Solving Linear Discrete Optimization Problems", JACM, 17: 2, April 1970, pp. 303-313.

67. Rovner, Paul D., An Investigation into Paging a Software-Simulated Associative Memory System, Document No. 40.10.90, Univ. of Calif., Berkeley, 1966.

68. Rovner, P. D. and J. A. Feldman, "The LEAP Language and Data Structure", Proc. IFIP Cong (1968), pp. 579-585.

69. Salton, Gerard, "Manipulation of Trees in Information Retrieval", CACM, 5: 2, February 1962, pp. 103-114.

70. Sammon, J. W., Jr., "A Nonlinear Mapping for Data Structure Analysis", IEEE Trans. Computers, C-18: 5, May 1969, pp. 401-409.

71. Sibley, Edgar H., et al, "Graphical Systems Communication: An Associative Memory Approach", Proc. FJCC (1968), pp. 545-555.

72. Sutherland, William R., "The CORAL Language and Data Structure", Excerpt from Doctoral Thesis, MIT Lincoln Laboratory, Lexington, Massachusetts, 1966.

73. Symes, L., "Manipulation of Data Structures in a Numerical Analysis Problem Solving System - NAPSS", Proc. SJCC (1970), pp. 157-164.

74. Toronto, A. F. et al, "Evaluation of a Computer Program for Diagnosis of Congenital Heart Disease", Prog. Cardiovascular Diseases, 5: 4, January 1963, pp. 362-377.

75. University of Michigan, "MTS: Michigan Terminal System", Volumes I, II, and III, University of Michigan Publication Distribution Services, February 1971.

76. Warner, H. R. et al, "A Mathematical Approach to Medical Diagnosis", JAMA, 177: 3, July 1961, pp. 177-183.

77. Warner, H. R. et al, "Experience with Bayes Theorem for Computer Diagnosis of Congenital Heart Disease", Annals NY Acad. Sciences, 115, 1964, pp. 558-567.

78. Weizenbaum, J., "Knotted List Structures", CACM, 5: 3, March 1962, pp. 161-165.

79. Weizenbaum, J., "Symmetric List Processor", CACM, 6: 9, September 1963, pp. 524-536.

80. Williams, Robin, "A Survey of Data Structures for Computer Graphics Systems", Computing Surveys, 3: 1, March 1971, pp. 1-21.

81 Woodward, P. M. and D. P. Jenkins, "Atoms and Lists", Computer Journal, April 1961, pp. 47-53.